PACKT PUBLISHING



掌握和使用正确的数据可视化方法

Python 数据可视化编程实战

Python Data Visualization Cookbook

[爱尔兰] Igor Milovanović 著
颛清山 译

人民邮电出版社
POSTS & TELECOM PRESS

目　录

Python数据可视化编程实战

[爱尔兰]lgor Milovanović　著
颛清山　译

人民邮电出版社

北京

图书在版编目（**CIP**）数据

Python数据可视化编程实战/（爱尔兰）米洛万诺维奇（Milovanovic,I.）著；颛清山译.--北京：人民邮电出版社，2015.5

ISBN　978-7-115-38439-3

Ⅰ.①P…　Ⅱ.①米…②颛…　Ⅲ.①软件工具—程序设计　Ⅳ.①TP311.56

中国版本图书馆CIP数据核字（2015）第057566号

◆著　[爱尔兰]Igor Milovanović

译　颛清山

责任编辑　陈冀康

责任印制　张佳莹　焦志炜

◆人民邮电出版社出版发行　　北京市丰台区成寿寺路11号

邮编　100164　　电子邮件　315@ptpress.com.cn

网址　http://www.ptpress.com.cn

三河市海波印务有限公司印刷

◆开本：800×1000　1/16

印张：16.75

字数：318千字　　2015年5月第1版

印数：1-3000册　　2015年5月河北第1次印刷

著作权合同登记号　图字：01-2013-9037号
定价：49.00元
读者服务热线：（**010**）**81055410**　印装质量热线：（**010**）**81055316**
反盗版热线：（**010**）**81055315**

内容提要

本书是一本使用 Python 实现数据可视化编程的实战指南，介绍了如何使用 Python 最流行的库，通过60余种方法创建美观的数据可视化效果。

全书共8章，分别介绍了准备工作环境、了解数据、绘制并定制化图表、学习更多图表和定制化、创建 3D 可视化图表、用图像和地图绘制图表、使用正确的图表理解数据以及更多的matplotlib知识。

本书适合那些对Python编程有一定基础的开发人员阅读，可以帮助读者从头开始了解数据、数据格式、数据可视化，并学会使用Python可视化数据。

译者序

图形可视化是展示数据的一个非常好的手段，好的图表自己会说话。毋庸多言，在Python的世界里，matplotlib是最著名的绘图库，它支持几乎所有2D绘图和部分3D绘图，被广泛地应用在科学计算和数据可视化领域。但是介绍matplotlib的中文书籍很少，大部分书籍只是在部分章节中提到了matplotlib的基本用法，因此在内容和深度上都力有不逮。本书则是一本专门介绍matplotlib的译著。

matplotlib 是一个开源项目，由 John Hunter 发起。关于 matplotlib 的由来，有一个小故事。John Hunter 和他研究癫痫症的同事借助一个专有软件做脑皮层电图分析，但是他所在的实验室只有一份该电图分析软件的许可。他和许多一起工作的同事不得不轮流使用该软件的硬件加密狗。于是，John Hunter 便有了开发一个工具来替代当前所使用的软件的想法。当时MATLAB被广泛应用在生物医学界中，John Hunter等最初是想开发一个基于MATLAB的版本，但是由于MATLAB的一些限制和不足，加上他本身对Python非常熟悉，于是就有了matplotlib的诞生。

所以，无论从名字上，还是从matplotlib提供的函数名称、参数及使用方法都与MATLAB非常相似。对于一个MATLAB开发人员，使用起来会相当得心应手。即使对不熟悉MATLAB的开发人员（譬如我），对其函数的使用也能够一目了然，而且matplotlib有着非常丰富的文档和实例，加上本书的介绍，学习起来将会非常轻松。

matplotlib 命令提供了交互绘图的方式，在 Python 的交互 shell 中，我们可以执行matplotlib命令来实时地绘制图形并对其进行修改。生成的

图像可以保存成许多格式，这取决于其所使用的后端，但绝大多数后端都支持如png、pdf、ps、eps和svg等格式。

在之前的项目中，我使用Python的Locust工具进行性能测试，该工具非常出色，然而在对获取到的性能数据的分析上，没有提供太多的功能。于是我决定使用 matplotlib 进行性能数据的分析和可视化。从绘制最简单的柱状图、线形图，到引入散点图、直方图，我渐渐对matplotlib有了进一步的了解，也对它提供了如此强大的功能却又不失易用性而着迷。

虽然本书是一本cookbook，然而它并不仅仅局限在讲解如何绘制各种图形上，更重要的是，本书让我们了解了如何用正确的图形把数据可视化出来，也就是“do the right thing in the right way”。在翻译本书的过程中，我意识到，如果我当时手头有这么一本书，将会少走不少弯路。本书包括了非常多的图形介绍以及丰富的示例，我相信，读完本书以后，读者将能应对各种常见的数据可视化问题。

在这里，我要特别感谢一下我的妻子董秋影，在精神和专业知识上她都给予了我莫大的帮助，没有她就没有这本译稿的完成。她从事医疗图像算法工作，对各种图形和算法以及MATLAB都有很深的了解，本书的每一章都经过了她认真的审阅校对。此外，感谢彭明伟先生完成了第1章的初稿翻译工作。最后，感谢人民邮电出版社陈冀康老师专业细心地审核，和陈老师合作很轻松、很开心。

由于译者水平有限，错误和失误在所难免，如有任何意见和建议，请不吝指正，我将感激不尽。我的邮箱：zhuanqingshan@163.com。

颛清山

2014年12月于北京

作者简介

Igor Milovanović是一个在Linux系统和软件工程领域有深厚背景的经验丰富的开发人员。具备创建可扩展数据驱动分布式富软件系统的技术。

他是一个高性能系统设计的布道者，对软件架构和开发方法论有着浓厚的兴趣。他一直坚持倡导促进高质量软件的方法论，如测试驱动开发、一键部署和持续集成。

他也拥有坚实的产品开发知识。拥有领域经验知识，并参加过官方培训，他能够在业务和开发人员之间很好地传递业务知识和业务流程。

非常感谢我的未婚妻，她允许我把大量的时间花费在工作上而没有陪伴她，并在我无休止地谈论本书时甘愿做一个热心的听众。我也想感谢我的哥哥，他一直是我坚强的后盾。感谢我的父母，给予我各种发展自己的空间，让我成为今天的自己。

如果没有开发Python、matplotlib和所有本书中使用的库的开源社区的巨大能量，我不可能写出这本书。我深深地感谢所有这些项目背后的人们。感谢你们！

评阅者简介

Tarek Amr 从东安格里亚大学获得了数据挖掘和信息检索专业的研究生学位。他在软件开发领域有近 10 年的经验。自从 2007 年开始，他一直在 Global Voices Online (GVO)义务工作，目前他是埃及 Open Knowledge Foundation (OKFN)的大使。他热衷于开放数据、Government 2.0、数据可视化、数据新闻、机器学习和自然语言处理。

Tarek的Twitter账号是@gr33ndata，主页是http://tarekamr.appspot.com/。

Jayesh K. Gupta是Matlab Toolbox for BiclusteringAnalysis (MTBA)的首席开发人员。他目前是一名IIT Kanpur的在读研究生和研究员。他的兴趣是模式识别。他对基础科学也有浓厚的兴趣，认为它们是自然界中的模式分析工具。来到IIT之后，他看到这种分析是如何借助机器学习算法广泛应用在各种不同的应用程序中的。他相信通过机器智能来强化人类的想法是增进人类知识的最好方式之一。他是一个长期的技术爱好者和自由软件的布道者。他的网名是rejuvyesh。他也是一名狂热的读者，从Goodreads可以获得他读过的书籍的信息。从Bitbucket和GitHub可以找到他的项目。所有的链接都在http://home.iitk.ac.in/～jayeshkg/上，也可以通过a2z.jayesh@gmail.com联系他。

Kostiantyn Kucher 出生在乌克兰敖德萨。2012 年他在敖德萨国立理工大学获得了计算机科学专业的硕士学位。他使用Python、Matplotlib和PIL从事机器学习和图像识别的工作。

目前，Kostiantyn是一名计算机科学专业信息可视化方向的博士研

究生。他在Andreas　Kerren博导的指导下，在瑞典林奈大学计算机科学系的ISOVIS小组进行研究。

Kenneth Emeka Odoh从事高级的数据可视化技术研究工作。他的研究兴趣是通过可视的线索指导用户得出研究结果的探索性研究。

Kenneth 精通 Python 编程。2012 年他曾在芬兰的 Pycon 大会做演讲，主题是 Django中的数据可视化。

他目前是加拿大里贾纳大学的一名研究员，通晓多种编程语言，有C、C++、Python和Java的应用开发经验。

编写代码之余，Kenneth还参加了坎皮恩学院圣歌合唱团。

前言

最好的数据是我们能看到并理解的数据。作为一个开发人员，我们想创造并构建出最全面且容易理解的可视化图形。然而这并非总是很简单，我们需要找出数据，读取它、清理它、揣摩它，然后使用恰当的工具将其可视化。本书通过简单（和不那么简单）直接的方法解释了如何读取、清理和可视化数据的流程。

本书对怎样读取本地数据、远程数据、CSV、JSON以及关系型数据库中的数据，都进行了讲解。

通过matplotlib，我们能用一行简单的Python代码绘制出一些简单的图表，但是进行更高级的绘图还需要除 Python 之外的其他知识。我们需要理解信息理论和人类的审美学来生成最吸引人的可视化效果。

本书讲解在Python中使用matplotlib绘图的一些练习、使用情况，以及对于不同图表特性应该使用的方法的一些最佳实践。

本书的写作及代码开发均基于 Ubuntu 12.03，使用了 Python 2.7、IPython 0.13.2、virtualenv 1.9.1、matplotlib 1.2.1、NumPy 1.7.1 和 SciPy 0.11.0。

本书涵盖内容

第1章，准备工作环境，包括一些安装方法，以及如何在你的平台上安装所需的Python包和库的一些建议。

第2章，了解数据，介绍通用的数据格式，以及如何读写，如CSV、JSON、XSL或者关系型数据库。

第3章，绘制图表及定制化，着手绘制简单的图表并介绍图表的定

制化。

第4章，学习更多图表和定制化，继续上一章内容，介绍更多的高级表格和网格定制化。

第5章，3D可视化，介绍三维数据的可视化，如3D柱状图、3D直方图，以及matplotlib动画。

第6章，用图像和地图绘制图表，涵盖图像处理、在地图上投射数据，以及创建CAPTCHA测试图像。

第7章，使用正确的图表理解数据，涵盖一些更高级绘图技术的讲解和方法，如频谱图和相关性。

第8章，更多的matplotlib知识，介绍一些图表如甘特图、箱线图，并且介绍如何在matplotlib中使用LaTeX渲染文本。

准备工作

学习本书时，需要你在自己的操作系统上安装 Python2.7.3 或最新版本。本书使用Ubuntu12.03系统上的默认Python版本（2.7.3）。

本书中用到的另一个软件包是 IPython，它是一个交互式的 Python环境，功能非常强大、灵活。可以通过基于 Linux 平台的包管理工具或者用于 Windows 和 Mac OS 系统的预安装文件安装。

一般来说，如果你对于Python安装和相关软件安装不熟悉，强烈推荐你使用预打包的Python 科学发行包如 Anaconda、Enthought Python 发行包或者 Python（X,Y）进行安装。

其他所需的软件主要包括Python包，可全部通过Python安装管理器pip进行安装。pip本身通过Python的easy_install安装工具安装。

谁适合阅读本书

本书是为那些通常已经了解Python编程的开发人员编写的。如果你听说过数据可视化但又不知道从何入手，本书会从头开始指导你了解数据、数据格式、数据可视化，以及如何使用Python可视化数据。

你需要知道一些一般的编程概念，如果你有编程经验，会非常有

用。然而，本书中的代码几乎是逐行讲解的。阅读本书不需要任何数学知识，书中介绍的每一个概念都有详细的讲解，并且提供了一些参考资料以供进一步的兴趣阅读。

约定

在本书中，不同的信息由一些不同风格的文字来区分。这里有一些文字风格的例子，以及它们的含义解释。

书中的代码文字显示如下："我们把小演示程序封装在DemoPIL类中，这样可以共享示例函数run_fixed_filters_demo的代码，并能很容易地对其进行扩展。"

代码块设置如下：

```
def _load_image(self, imfile):
    self.im = mplimage.imread(imfile)
```

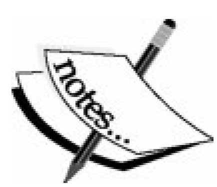
当我们想要让你关注代码块中的某一特定部分时，相关的行或元素将设置为粗体：

```
# tidy up tick labels size
all_axes = plt.gcf().axes
for ax in all_axes:
    for ticklabel in ax.get_xticklabels() + ax.get_yticklabels():
        ticklabel.set_fontsize(10)
```

所有的命令行输入或者输出的写法如下。

```
$ sudo python setup.py install
```

新术语和关键词将显示为粗体。例如，在屏幕上、菜单或对话框中的文字将会显示为："然后我们为火柴杆图设置一个标签和基线位置，默认值为**0**。"



警告或者重要的说明出现在这样的一个文本框中。



提示和技巧像这样显示。

读者反馈

欢迎读者向我们反馈意见。请让我们知道你对本书的看法——哪些是你喜欢或者不喜欢的。读者反馈对我们非常重要，可以帮我们完善一些你非常关心的内容。

如果给我们发送一般的反馈，可以简单地发电子邮件到feedback@packtpub.com，并请在消息标题中提及书名。

如果你对某个话题有经验，并且有兴趣写作或者想为一本书做贡献，请参考我们的作者指南www.packtpub.com/authors。

支持

既然你已经是Packt书籍的读者，我们有许多辅助材料可以帮助你从本书中得到最大的收获。

下载示例代码

可以在 http://www.packtpub.com 网站上你的账户中下载你所购买的所有图书的示例代码文件。如果你在其他地方购买本书，可以访问http://www.packtpub.com/support页面进行登记，文件会通过邮件直接发送给你。

勘误

尽管我们已经竭尽全力确保本书内容的准确，但是错误在所难免。如果你在书中发现了错误——可能是文本或者代码中的错误——如果你能把它报告给我们，我们将万分感谢。如此，这样可以减轻其他读者的痛苦，并且可以帮我们改进该书的后续版本。如果你找到任何错误，请访问http://www.packtpub.com/submit-errata并报告给我们，选择你的图

书，点击 errata submission form 链接，并加入你勘误的详细内容。一旦你的勘误通过验证，你的提交将被接受，勘误将会上传到我们的网站，或者添加到位于该标题的勘误部分的已有勘误列表中。可以在http://www.packtpub.com/support上选择你的标题来查看所有现有的勘误信息。

著作权侵害

互联网上的版权侵害是一个跨越所有媒介的持续的问题。在Packt，我们很认真地看待版权和许可保护。如果你不经意在互联网上得到了关于我们作品的任何形式的不合法的副本，请及时给我们提供其地址或者网站名称，以便我们及时补救。

请通过copyright@packtpub.com联系我们，同时请提供涉嫌侵权材料的链接。

非常感激你帮助保护我们的作者，让我们尽力提供更有价值的内容。

问题

如果你对本书有任何疑问，可以通过 questions@packtpub.com 联系我们，我们会竭尽全力提供帮助。

第1章 准备工作环境

本章包含以下内容。

- ◆ 安装matplotlib、NumPy和SciPy库
- ◆ 安装virtualenv和virtualenvwrapper
- ◆ 在Mac OSX上安装matplotlib
- ◆ 在 Windows 上安装 matplotlib
- ◆ 安装Python图像处理库（Python Imaging Library，PIL）
- ◆ 安装requests模块
- ◆ 通过代码设置matplotlib的参数
- ◆ 为项目设置matplotlib的参数

1.1 介绍

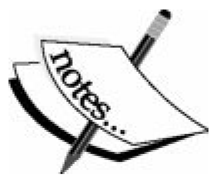
本章向读者介绍必备的工具类库，以及如何进行安装与配置。作为本书后续部分的基础知识，掌握这部分内容十分必要。如果你没有使用Python进行数据处理、图像处理以及数据可视化的经验，建议不要跳过本章。如略过本章，在需要安装配套工具软件或需要确定工程所支持的软件版本时，可返回本章阅读相关内容。

1.2 安装matplotlib、Numpy和Scipy库

本章介绍了matplotlib及其依赖的软件在Linux平台上的几种安装方法。

1.2.1 准备工作

这里假设你已经安装了Linux 系统且安装好了Python （推荐使用Debian/Ubuntu 或RedHat/SciLinux）。在前面提到的Linux系统发行版中，Python通常是默认安装的。如果没有，使用标准的软件安装方式安装Python也是非常简便的。本书假设你安装的Python版本为2.7或以上。



几乎所有的代码均可在 Python 3.3 及以上版本的环境下工作，但是因为大部分操作系统提供的Python版本仍然是2.7（甚至是2.6），本书代码基于Python 2.7 版本。这种基于 Python 版本的区别并不大，主要是在软件包版本和部分代码上存在差别（在Python3.3以上版本，请使用range方法替换xrang方法）。

本书也假设你知道如何使用操作系统软件包管理工具进行软件包的安装，以及知道如何使用命令行终端。

构建matplotlib运行环境，需要满足相关软件依赖。

Matplotlib的构建过程依赖NumPy、libpng和freetype软件包。要从源代码构建matplotlib，必须先要安装好NumPy库。读者可以访问http://www.numpy.org/了解安装NumPy库的方法（请安装1.4或以上版本，Python 3需要NumPy 1.5或以上版本）。



NumPy库提供处理大数据集的数据结构和数学方法。诸如元组、列表或字典等Python的默认数据结构同样可以很好地支持数据的插入、删除和连接。NumPy的数据结构支持“矢量”操作，使用简便，同时具有很高的执行效率。矢量操作在实现时充分考虑了大数据的需要，基于C语言的实现方式也保证了执行效率。

基于NumPy构建的SciPy库，是Python的标准科学计算和数学计算工具包，包含了大量的专用函数和算法。而大部分函数和算法源自著名的Netlib软件仓库（参见http://www.netlib.org），实际上是使用C语言和Fortran语言实现的。

安装NumPy库的步骤如下。

1.安装Python-NumPy软件包。

$ sudo apt-get install python-numpy

2.检查软件包版本。

$ python -c 'import numpy; print numpy.__version__'

3.安装所需的库。

◆ libpng 1.2：PNG 文件处理（依赖 zlib 库）。

◆ freetype 1.4+：处理 True type 字体。

$ sudo apt-get install build-dep python-matplotlib

如果使用RedHat或基于RedHat的Linux发行版（Fedora、SciLinux或CentOS），可以使用yum工具进行安装，方法与apt-get工具类似。

$ su -c 'yum-builddep python-matplotlib'

1.2.2 操作步骤

安装matplotlib及其依赖软件的方法有很多：从源代码安装，使用预

编译完成的二进制文件安装，通过操作系统软件包管理工具安装，或安装内置了matplotlib的python预打包发布版本。

使用包管理工具大概是最简单的安装方式。例如在Ubuntu系统中，在命令行终端中输入下面的命令即可。

```
# in your terminal, type:
$ sudo apt-get install python-numpy python-matplotlib python-scipy
```

如果读者期望使用最新特性，最好的选择是通过源代码进行安装。安装方式包含以下步骤：获取源代码、构建依赖库和参数配置、编译以及安装。

可以从代码托管站点www.github.com下载最新代码进行安装，操作步骤如下。

```
$ cd ～/Downloads/
$  wget  https://github.com/downloads/matplotlib/matplotlib/matplotlib-1.2. 0.tar.gz
$ tar xzf matplotlib-1.2.0.tar.gz
$ cd matplotlib-1.2.0
$ python setup.py build
$ sudo python setup.py install
```




下载示例代码

对于使用网站账户在 http://www.packtpub.com 上购买的所有 Packt 书籍，读者均可在网站上下载有关的代码示例。如果读者是从别处购得图书，可以访问网址（http://www.packtpub.com/support/），完成注册后，代码文件会发送到读者邮箱。

1.2.3 工作原理

从源代码安装 matplotlib, 使用了标准的 Python 发布工具 Distutils。安装过程需要提前安装依赖的软件包。关于使用标准的 Linux 包管理工具安装依赖软件的方法，可参考本节中关于准备工作的说明。

1.2.4 补充说明

根据数据可视化项目的需要，可能有必要安装额外的可选软件包。

无论你工作在什么项目上，IPython都是值得推荐的。IPython是一款交互式Python命令行工具。其提供的PyLab模式，已经导入了matplotlib库与相关软件包（例如NumPy和SciPy），可以直接使用相关库的功能。IPython 工具的安装与使用方法十分简单明了，读者可通过IPython 的官方网站查看相关细节。

1.3 安装virtualenv和virtualenvwrapper

如果同时工作在多个项目上，或是需要在不同项目间频繁切换，将所有的软件都安装在操作系统层级上也许不是一个好主意。当需要在不同系统（产品环境）上运行软件时，这种方式会带来问题。如果到此时才发现缺少特定的软件包，或是产品环境已经安装的软件包存在版本冲突，这将是非常痛苦的。为避免这种情况发生，可以选择使用virtualenv。

virtualenv 是由 Ian Bicking 创建的开放源代码项目。通过这个项目，开发人员可以把不同项目的工作环境隔离开，从而能够更容易地维护多种不同的软件包版本。

举例来说，Django 网站系统是基于 Django 1.1 和 Python 2.3 版本开发的，但与此同时，一个新项目要求必须基于Python2.6来开发。在笔者工作过的项目中，根据项目的需要同时使用多个版本的Python（以及相关软件包）的情况非常普遍。

virtualenv能够让我们很容易地在不同的运行环境之间切换。同时，如果需要切换到另外的机器或者需要在产品服务器（或客户的工作站主机）上部署软件，用 virtualenv 能够很容易地重新构建相同的软件包环境。

1.3.1 准备工作

若安装virtualenv，需要用到Python和pip。Pip是安装并管理Python软件包的工具，可以用它来代替 easy install 工具。本书中大部分的软件包都是用 pip 工具进行管理的。只需在终端中以root身份执行如下命

令，就可以很容易地完成pip的安装。

easy_install pip

virtualenv 本身已经相当不错了，然而如果配合 virtualenvwrapper，一切变得更加简单，并且组织多个虚拟环境的工作也会更加容易。virtualenvwrapper 的功能请参考 http://virtualenvwrapper.readthedocs.org/en/latest/#features。

1.3.2 操作步骤

安装virtualenv和virtualenvwrapper工具的步骤如下。

1.安装virtualenv和virtualenvwrapper。

$ sudo pip virtualenv

$ sudo pip virtualenvwrapper

创建保存虚拟环境的目录，并使用 export 导出为环境变量。

$ export VIRTENV=～/.virtualenvs

$ mkdir -p $VIRTENV

使用 source 命令调用（执行）shell 脚本来激活包装器

$ source /usr/local/bin/virtualenvwrapper.sh

创建一个虚拟环境

$ mkvirtualenv virt1

2.在virt1环境中安装matplotlib。

(virt1)user1:～$ pip install matplotlib

3.很有可能需要把以下代码添加到～/.bashrc中。

source /usr/loca/bin/virtualenvwrapper.sh

下面是一些有用和频繁使用的命令。

◆ mkvirtualenv ENV: 创建名为 ENV 的虚拟环境并激活。

◆ workon ENV: 激活先前创建的 ENV 虚拟环境。

- deactivate: 退出当前虚拟环境。

1.4 在Mac OS X上安装matplotlib

在 Mac OS X 上获取 matplotlib 最简便的方式是使用预打包的python 发布版本，例如Enthought Python Distribution (EPD)。读者可以直接访问 EPD 网站，下载安装操作系统对应的最新稳定版。

倘若EPD软件不满足要求，或者因为其他一些原因（如版本问题）而无法使用，也可以用手动（麻烦点）的方式安装Python、matplotlib和依赖软件。

1.4.1 准备工作

对于Apple在操作系统中没有安装的软件来说，Homebrew项目可以使安装过程更容易。实际上，Homebrew是基于Ruby和Git的，可以被自动下载和安装。软件安装顺序为：首先安装 Homebrew,之后安装Python，随后安装诸如 virtualenv 的工具软件，接下来安装matplotlib的依赖（NumPy和SciPy），最后安装matplotlib。接下来就开始吧。

1.4.2 操作步骤

1.在终端中输入并执行下面的命令。

```
ruby <(curl -fsSkL raw.github.com/mxcl/homebrew/go)
```

命令执行完成后，可以尝试用 brew update 或 brew doctor 命令来检查 brew 是否能够正常工作。

2.然后，将Homebrew目录添加到系统path环境变量中。这样，使用Homebrew安装的软件包能够获得比其他版本更高的优先级。打开～/.bash_profile文件（或者/Users/[your-user-name]/.bash_profile）并在文

件末尾添加以下代码。

```
export PATH=/usr/local/bin:$PATH
```

3.重新启动命令行终端使其加载新的 path 环境变量。之后，下面一行简单的代码就可以完成Python的安装。

```
brew install python --framework –universal
```

本命令同时也将安装Python所需的其他软件。

4.更新path环境变量（添加到同一行）。

```
export PATH=/usr/local/share/python:/usr/local/bin:$PATH
```

5.在命令行输入 python –version,检查 python 是否安装成功。

正常的话，会能够看到Python版本信息为2.7.3。

6.pip应该也已经安装完毕。如果还没有，可使用easy_install安装pip。

```
$ easy_install pip
```

7.这时，任何所需软件包的安装过程就变得非常简单了。例如，安装 virtualenv 和virtualenvwrapper。

```
pip install virtualenv
pip install virtualenvwrapper
```

8.是时候向一直以来的目标迈进了——安装matplotlib。

```
pip install numpy
brew install gfortran
pip install scipy
```



Mountain Lion 的用户需要安装 SciPy 的开发版（0.11），命令如下。

```
pip install -e git+https://github.com/scipy/scipy#egg=scipy- dev
```

9.检查安装是否成功。启动Python并执行以下命令。

```
import numpy
print numpy.__version__
import scipy
print scipy.__version__
quit()
```

10.安装matplotlib。

```
pip install matplotlib
```

1.5 在Windows上安装matplotlib

在本节中，我们将演示如何安装Python和matplotlib。假设系统中没有预先安装Python。

1.5.1 准备工作

在Windows上安装matplotlib有两种方式。较简单的方式是安装预打包的Python环境，如EPD、Anaconda和Python(x,y)。这是本书推荐的安装方式，尤其对于初学者来说更是如此。

第二种方式，是使用预编译的二进制文件来安装matplotlib和依赖软件包。需要注意安装的NumPy和SciPy的版本，因为并非所有的版本都与最新版matplotlib二进制文件相互兼容，这势必会给整个安装过程带来一些困难。这种安装方法也有自身的优势。如果想要获取最新功能，即使功能还未正式发布，仍然能够通过编译matplotlib或某软件库的某个特定版本来使用它。

1.5.2 操作步骤

要安装免费或商业Python科学发布版，按照项目网站上提供的步骤可以很容易安装成功，这也是推荐使用的方式。

如果单纯使用matplotlib,不期望面对Python和依赖软件包版本所带来的困扰，可以考虑使用 Enthought Python Distribution(EPD)发布版。使用matplotlib 所需的预打包库和所有必须的依赖软件（SciPy、NumPy、IPython以及更多的其他软件包），均已包含在EPD发布版中。

matplotlib 以及与本书内容相关的软件，都可以使用常规的

Windows Installer 安装文件（*.exe）方式进行安装。

Python(x,y)（http://code.google.com/p/pythonxy/）是针对 Windows 32 位系统的免费科学计算项目，其中包含了 matplotlib 需要使用的依赖文件，它是在 Windows 系统上安装matplotlib 的一种非常简单（而且是免费的）的方式。因为 Python(x,y)和 Python 模块安装器相互兼容，可以很容易地在Python(x,y)基础上扩展安装其他Python库。在安装Python(x,y)之前，系统应该没有安装Python。

下面简短地说明一下如何使用预编译的Python、NumPy、SciPy和matplotlib二进制文件进行matplotlib的安装。首先，下载官方的MSI安装文件安装对应平台（x86或x86-64）的标准Python程序。之后，下载NumPy和SciPy的官方二进制文件并安装它们。在正确安装NumPy和SciPy之后，就可以下载最新稳定版matplotlib二进制安装文件并按照官方说明进行安装了。

1.5.3 补充说明

请注意，在Windows安装文件中matplotlib的示例相当有限。如果想尝试使用示例程序，可以下载并参考matplotlib源文件包中的examples子目录。

1.6 安装图像处理工具：Python图像库（PIL）

Python图像库（PIL）为Python提供了图像处理能力。PIL支持的文件格式相当广泛，在图像处理领域提供了相当强大的功能。

快速数据访问、点运算（point operations）、滤波（filtering）、图像缩放、旋转、任意仿射转换（arbitrary affine transforms）是 PIL 中一些应用非常广泛的特性。例如，图像的统计数据即可通过histogram方法获得。

PIL 同样可以应用在其他方面，如批量处理、图像压缩、生成缩略图、图像格式转换以及图像打印。

PIL 可以读取多种图像格式，而图像写入支持的格式范围限定在图像交换和展示方面最通用的格式（有意为之）。

1.6.1 操作步骤

最容易也是最值得推荐的方式，是通过操作系统平台的包管理工具进行安装。

在Debian/Ubuntu系统中安装的命令如下。

```
$ sudo apt-get build-dep python-imaging
$ sudo pip install http://effbot.org/downloads/Imaging-1.1.7.tar.gz
```

1.6.2 安装过程说明

我们通过apt-get系统工具安装PIL所需的所有依赖软件，并通过pip安装PIL的最新稳定版本。一些老版本的Ubuntu系统通常不会提供PIL的

最新发布版本。

在RedHat/SciLinux系统中，安装命令如下。

```
# yum install python-imaging
# yum install freetype-devel
# pip install PIL
```

1.6.3 补充说明

有一个专门针对 PIL 编写的在线手册。读者可以访问 http://www.pythonware. com/library/pil/handbook/index.htm 进行阅读，或是下载 PDF 版本：http://www.pythonware. com/media/data/pil-handbook.pdf。

Pillow 是一个 PIL 分支，它的主要目的是解决安装过程中的一些问题。Pillow 很容易安装，其网址为http://pypi.python.org/pypi/Pillow。

在Windows平台上，也可使用二进制安装文件安装PIL。从 http://www.pythonware. com/products/pil/下载.exe安装文件，执行该文件将安装 PIL 到 Python 的 site-packages目录。

如果需要在虚拟环境下使用 PIL，可手动将 PIL.pth 文件和位于 C:\Python27\Lib\site-packages下的PIL目录复制到virtualenv的site-packages目录下。

1.7 安装requests模块

我们需要的大部分数据都可以通过HTTP或类似协议获得，因此我们需要一些工具来实现数据访问。Python的requests库能让这部分工作变得轻松起来。

虽然Python提供的urllib2模块提供了访问远程资源的能力以及对HTTP协议的支持，但使用该模块完成基础任务的工作量还是很大的。

Request模块提供新的API，减轻了使用Web服务的痛苦，使其变得更直接。Requests封装了很多 HTTP 1.1 的内容，仅在需要实现非默认行为的情况下才需要暴露相关内容。

1.7.1 操作步骤

安装requests模块最好的方式是使用pip。安装命令如下。

```
$ pip install requests
```

也可以在virtualenv虚拟环境中执行安装命令，如果并不是所有项目都需要requests，或是不同的项目需要使用不同版本的requests。

为了更快地理解requests的功能，下面是一个使用requests的小例子。

```
import requests
r = requests.get('http://github.com/timeline.json')
print r.content
```

1.7.2 requests 使用说明

在本例中，我们向 www.github.com 站点的 URI 发送 HTTP GET 请

求，以 JSON格式返回了 GitHub 网站的活动时间表（也可以通过访问https://github.com/timeline 得到HTML版本的活动时间表）。在成功读取HTTP响应后，对象r包含了HTTP响应内容以及其他属性信息（HTTP状态码、cookies、HTTP头元数据，甚至包括当前响应所对应的请求信息）。

1.8 在代码中配置matplotlib参数

matplotlib库提供了强大的绘图功能，是本书用的最多的Python库。在其配置文件即.rc文件中，已经为大部分属性设定了默认值。本节会介绍如何通过应用程序代码修改matplotlib的相关属性值。

1.8.1 准备工作

如前所述，matplotlib配置信息是从配置文件读取的。在配置文件中可以为matplotlib的几乎所有的属性指定永久有效的默认值。

1.8.2 操作步骤

在代码执行过程中，有两种方式更改运行参数：使用参数字典（rcParams）或调用matplotlib.rc()命令。第一种方式中，可以通过rcParams字典访问并修改所有已经加载的配置项；第二种方式中，可以通过向 matlotlib.rc()传入属性的关键字元组来修改配置项。

如果需要重置动态修改后的配置参数，可以调用matplotlib.rcdefaults()将配置重置为标准设置。

下面两段代码演示了之前介绍的功能。

使用matplotlib.rcParams的例子。

```
import matplotlib as mp
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.color'] = 'r'
```

使用matplotlib.rc()函数调用的例子。

```
import matplotlib as mpl
```

```
mpl.rc('lines', linewidth=2, color='r')
```

上面两个例子具有相同的语义。第二个例子中，我们设定后续的所有图形使用的线条宽度为2个点。第一个例子中的最后一条语句表明，语句之后的所有线条的颜色均为红色，除非用本地设置覆盖它，请看下面的例子。

```
import matplotlib.pyplot as plt
import numpy as np
t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2 * np.pi * t)
# make line red
plt.rcParams['lines.color'] = 'r'
plt.plot(t,s)
c = np.cos(2 * np.pi * t)
# make line thick
plt.rcParams['lines.linewidth'] = '3'
plt.plot(t,c)
plt.show()
```

1.8.3 代码解析

首先，为了绘制正弦、余弦曲线，需要导入matplotlib.pyplot和NumPy模块。在绘制第一个图像之前，通过 plt.rcParams['lines.color']= 'r'语句显式地设置线条颜色为红色；接下来，对于第二个图像（余弦曲线），通过语句plt.rcParams['lines. linewidth'] = '3'显式地设定线宽为 3 个点。

如果需要重置设置，需要调用matplotlib.rcdefaults()方法。

1.9 为项目设置matplotlib参数

本节介绍matplotlib使用的各种配置文件的位置，以及使用这些配置文件的意义。同时还将介绍配置文件中的具体配置项。

1.9.1 准备工作

如果不想在每次使用 matplotlib 时都在代码开始部分进行配置（像前一节我们做的那样），就需要为不同的项目设定不同的默认配置项。本节将介绍如何做到这一点。这种配置方式使得配置项与代码分离，从而使代码更加整洁。此外，你可以很容易在同事间甚至项目间分享配置模板。

1.9.2 配置方法

假设一个项目对于matplotlib的特性参数总会设置相同的值，就没有必要在每次编写新的绘图代码时都进行相同的配置。取而代之的，应该是在代码之外，使用一个永久的文件设定matplotlib参数默认值。

通过 matplotlibrc 来配置文件，matplotlib 提供了对这种配置方式的支持。在matplotlibrc文件中包含了绝大部分可以变更的属性。

1.9.3 配置过程说明

配置文件可能存在于三个不同的位置，而它们的位置决定了它们的应用范围。这三个位置分别说明如下。

◆ 当前工作目录：即代码运行的目录。在当前目录下，可以为目录所包含的当前项目代码定制matplotlib配置项。配置文件的文件名是

matplotlibrc。

◆ 用户级.matplotlib/matplotlibrc 文件(Per user .matplotlib/matplotlibrc)：通常是在用户的$HOME 目录下（在 Windows 系统中，也就是 Documents and Settings 目录）。可以用 matplotlib.get_configdir()命令来找到当前用户的配置文件目录。请参考随后的命令示例。

◆ 安装级配置文件（Per installation configuration file）：通常在 python的site-packages目录下。这是系统级配置，不过在每次重新安装 matplotlib后，配置文件会被覆盖。因此如果希望保持持久有效的配置，最好选择在用户级配置文件中进行设置。对于笔者来说，目前对本配置文件的最佳应用方式，是将其作为默认配置模板。如果在用户级配置文件已经比较混乱，或者需要为新项目做全新配置时，可以基于该配置文件进行设置。

在shell中运行下面的命令，即可打印出配置文件目录的位置：

```
$ python -c 'import matplotlib as mpl; print mpl.get_configdir()'
```

配置文件包括以下配置项。

◆ axes：设置坐标轴边界和表面的颜色、坐标刻度值大小和网格的显示。

◆ backend：设置目标输出 TkAgg和 GTKAgg。

◆ figure：控制 dpi、边界颜色、图形大小和子区（subplot）设置。

◆ font：字体集（font family）、字体大小和样式设置。

◆ grid：设置网格颜色和线型。

◆ legend：设置图例和其中文本的显示。

◆ line：设置线条（颜色、线型、宽度等）和标记。

◆ patch：是填充 2D 空间的图形对象，如多边形和圆。控制线宽、颜色和抗锯齿设置等。

◆ savefig：可以对保存的图形进行单独设置。例如，设置渲染的文

件的背景为白色。

◆ text：设置字体颜色、文本解析（纯文本或 latex 标记）等。

◆ verbose：设置 matplotlib 在执行期间信息输出，如 silent、helpful、debug 和debug-annoying。

◆ xticks 和 yticks：为 x、y 轴的主刻度和次刻度设置颜色、大小、方向，以及标签大小。

1.9.4 补充说明

如果你想了解前面提到的（和我们没有提到的）每个设置的详细信息，最好的方式是访问 matplotlib 项目的网站，那里提供了最新的 API 文档。如果需要获得进一步帮助，可以在用户和开发邮件组留言。本书最后还提供了一些有用的在线资源。

第2章 了解数据

在本章中，我们会介绍以下内容。

◆ 从 CSV 文件导入数据
◆ 从 Microsoft Excel 文件导入数据
◆ 从定宽数据文件导入数据
◆ 从制表符分隔的文件导入数据
◆ 从 JSON 数据源导入数据
◆ 导出数据到 JSON、CSV 和 Excel
◆ 从数据库导入数据
◆ 清理异常值
◆ 读取大块数据文件
◆ 读取流数据源
◆ 导入图像数据到 NumPy 数组
◆ 生成可控的随机数据集合
◆ 真实数据的噪声平滑处理

2.1 简介

本章涵盖了导入和导出各种格式数据的基本知识。除此之外，还包括清理数据的方式，比如值的归一化处理、缺失数据的添加、实时数据检查以及一些类似的技巧，以便正确地准备数据来进行可视化。

2.2 从CSV文件导入数据

在本节中，我们将处理每个人都能接触到的最常用的文件格式——CSV。顾名思义，CSV是指逗号分隔的值（文件中还包括一个文件头，也是以逗号分隔的）。

Python中有个csv模块支持读写各种方言格式的CSV文件。方言是很重要的，因为没有一个统一的CSV标准，不同的应用实现CSV的方式略有不同。当看到文件内容的时候，你往往就能很容易地辨认出文件使用的是哪种方言。

2.2.1 准备工作

在本节中，我们把ch02-data.csv文件的内容用作示例数据，你可以把它下载到本地。

我们假定示例数据文件和读取数据文件的代码在相同目录下。

2.2.2 操作步骤

下面的示例代码解释了如何从CSV文件导入数据，步骤如下。

1.打开ch02-data.csv文件。

2.首先读取文件头。

3.然后读取剩余行。

4.当发生错误时抛出异常。

读取完所有内容后，打印文件头和其余所有行。

```
import csv
filename = 'ch02-data.csv'
```

```
data = []
try:
    with open(filename) as f:
        reader = csv.reader(f)
    header = reader.next()
    data = [row for row in reader]
except csv.Error as e:
    print "Error reading CSV file at line %s: %s" % (reader.line_num, e)
    sys.exit(-1)
if header:
    print header
    print '=================='
for datarow in data:
    print datarow
```

2.2.3 工作原理

首先，导入csv模块以便能访问所需的方法。然后，用with语句打开数据文件并把它绑定到对象f。不必操心在操作完资源后去关闭数据文件，with语句的上下文管理器会帮助处理。这在操作资源型文件时非常方便，因为它能确保在代码块执行完毕后资源会被释放掉（比如关闭文件）。

然后，用csv.reader()方法返回reader对象，通过该对象遍历所读取文件的所有行。在这里，每行内容不过是一个值列表，在循环中被打印出来。

文件的第一行是文件头，用来描述文件中每列的数据，在读取时多少有些不同。文件头并不是必需的，有些CSV文件就不带文件头，但是

它们确实是提供数据集合的最小元数据信息的一个不错的方式。然而，有时候会碰到用分隔的文本或者仅用作元数据的CSV文件，来描述数据格式和附加数据的情况。

此时只能打开文件来看看第一行是数据头还是数据（例如查看文件的前几行）。这在Linux系统上用bash命令如head可以很容易做到，格式如下所示。

```
$ head some_file.csv
```

在遍历数据时，我们把第一行存储为文件头，把其他行添加到数据列表中。

读取文件时一旦出了问题，csv.reader()方法会生成错误信息。为了能帮助用户发现问题，可以捕获这些错误信息并给用户打印出有用的信息。

2.2.4 补充说明

如果想了解csv模块的来龙去脉，可以看一下PEP文档中的《CSV文件API》，参见http://www.python.org/dev/peps/pep-0305/。

如果想加载大数据文件，明智的做法通常是使用一些著名的库如NumPy的loadtxt()方法，这个方法可以很好地处理CSV大数据文件。

基本用法非常简单，如下面的代码段所示。

```
import numpy
data=numpy.loadtxt('ch02-data.csv',dtype='string', delimiter=',')
```

值得注意的是，为了能让NumPy正确地分隔数据，需要定义分隔符。numpy.loadtxt()方法比类似的numpy.genfromtxt()方法要快一些，但是后者能更好地处理缺失数据，而且在处理已加载文件的某些列时，可以使用一些方法来做些额外的事情。

目前，在 Python 2.7.x 版本中，csv模块不支持 Unicode 编码，必须把读取的数据显式地转换成可打印的UTF-8或者ASCII编码。官方的Python CSV文档提供了一些解决数据编码问题的很好的示例。

Python3.3及后续版本默认支持Unicode编码，不存在此类问题。

2.3 从Microsoft Excel文件中导入数据

虽然 Microsoft Excel 支持一些画图操作，但如果需要更加灵活和强大的可视化效果，就需要把数据从表单中导出到Python中以备将来之需。

从Excel文件导入数据的通常做法是把数据从Excel中导出到 CSV格式的文件中，然后用上节中提到的方法使用Python从CSV文件中导入数据。如果只有一两个文件（并且安装了 Microsoft Excel 或者OpenOffice.org），事情就相当简单。但是如果想自动化地对大量文件进行数据管道处理（作为数据连续处理流程的一部分），那么手动把每个Excel文件转换成CSV文件的做法就行不通了。因此，我们需要一种方法来读取Excel文件。

通过www.python-excel.org项目提供的软件包，Python可以很好地支持Excel文件的读写操作。对读操作和写操作的支持是通过不同模块实现的，而且是平台无关的。换言之，我们不必为了读取Excel文件而必须要在Windows平台上工作。

Microsoft Excel 文件格式随着时间发生着变化，不同的 Python 库对其都有相应的支持。在写作本书时，XLRD最新的稳定版本是0.90，它已经支持读取.xlsx文件了。

2.3.1 准备工作

首先，我们需要安装所需的模块，在这个例子中我们将使用xlrd模块。我们将用pip在虚拟环境中安装此模块。

$ mkvirtualenv xlrdexample

```
(xlrdexample)$ pip install xlrd
```

安装完毕后，我们将用ch02-xlsxdata.xlsx示例文件做演示。

2.3.2 操作步骤

接下来的示例代码将展示如何从已知的Excel文件中读取一个样本数据集合。

1.打开文件的工作簿。

2.根据名称找到工作表。根据行数（nrows）和列数（ncols）读取单元格的内容。

3.因为只是用作演示，本例仅打印出了读取的数据集合。

```
import xlrd
file = 'ch02-xlsxdata.xlsx'
wb = xlrd.open_workbook(filename=file)
ws = wb.sheet_by_name('Sheet1')
dataset = []
for r in xrange(ws.nrows):
    col = []
    for c in range(ws.ncols):
        col.append(ws.cell(r, c).value)
    dataset.append(col)
from pprint import pprint
pprint(dataset)
```

2.3.3 工作原理

让我们试着解释一下 xlrd 模块使用的简单对象模型。在最上层是一个包含一个或多个工作表（xlrd.sheet.Sheet）的工作簿（Python 类

xlrd.book.Book）。每个工作表有一个单元格对象（xlrd.sheet.Cell），我们能从单元格中将值读取出来。

通过调用open_workbook()方法，我们从文件中加载了一个工作簿，并返回一个xlrd.book 实例。Book 实例包含了一个工作簿的所有信息，如工作表单。通过调用sheet_by_name()方法可以访问指定的工作表，如果需要所有的工作表，可以调用sheets()方法。sheets()方法返回一个xlrd.sheet.Sheet 实例的列表。xlrd.sheet.Sheet类有行和列属性，我们能通过这些属性来指定循环的范围，并通过调用cell()方法来访问工作表中的每个特定的单元格。虽然有一个xrld.sheet.Cell类，但并不需要直接使用它。

请注意，日期是以浮点数而不是以某个日期类型存储的。但是，xlrd 模块有能力检查数据的值，并推断出数据值实际上是否为一个日期。这样，我们就能通过检查单元格类型来得到 Python date 对象。如果数字的字符串像日期，xlrd 模块将返回 xlrd.XL_CELL_DATE作为单元格类型。这里用一段代码来说明这点：

```
from datetime import datetime
from xlrd import open_workbook, xldate_as_tuple
...
cell = sheet.cell(1, 0)
print cell
print cell.value
print cell.ctype
if cell.ctype == xlrd.XL_CELL_DATE:
    date_value = xldate_as_tuple(cell.value, book.datemode)
    print datetime(*date_value)
```

这个日期字段还有些问题。因此，如果需要针对日期做大量的工作，请参见官方文档和邮件列表。

2.3.4 补充说明

xlrd 模块的一个非常好的特性是它能按照需要仅加载文件的部分内容到内存中。open_workbook方法有一个on_demand参数，在调用时把它置为True，工作表就能按需加载了。例如：

book = open_workbook('large.xls', on_demand=True)

本节没有提到 Excel 文件的写操作。一部分原因是后面会有单独的一节去讲述它，另一部分原因是Excel的写操作需要另一个不同的模块——xlwt来完成。你能从本章的“导出数据到JSON、CSV和Excel”一节获得更多的信息。

如果需要一些在前面介绍的例子和模块中没有涉及的特定用法，PyPi 上有一个操作工作表的其他一些Python模块的列表，也许能对你有帮助，请访问http://pypi.python.org/pypi?:action=browse&c=377。

2.4 从定宽数据文件导入数据

事件的日志文件和基于时间序列的文件是数据可视化中最常见的数据源。有时候，可以以制表符分隔数据这种CSV方言来读取它们，但有时它们不是通过任何特殊字符分隔的。实际上，这些文件中的字段是有固定宽度的，我们能通过格式来匹配并提取数据。

一种做法是逐行读取文件，然后用字符串操作方法把字符串分割成独立的部分。这种做法比较直接，如果性能不是问题的话可以作为首选。

如果性能更重要，或者要解析的文件非常大（几百兆字节），用Python中的struct模块（http://docs.python.org/library/struct.html）能提升性能，因为这个模块是用C语言而不是Python实现的。

2.4.1 准备工作

因为struct模块是Python标准库的一部分，所以不必安装额外的软件来完成本节的内容。

2.4.2 操作步骤

我们将会使用一个预先生成的数据集合，其中有一百万行定宽数据记录。样本数据格式如下：

```
...
207152670 3984356804116 9532
427053180 1466959270421 5338
316700885 9726131532544 4920
```

```
138359697 3286515244210 7400
476953136 0921567802830 4214
213420370 6459362591178 0546
...
```

这个数据集合是通过代码生成的，代码文件ch02-generate_f_data.py可以在本章的代码库中找到。

现在可以读取数据了。示例代码如下，步骤如下。

1.指定要读取的数据文件。

2.定义数据读取的方式。

3.逐行读取文件并根据格式把每行解析成单独的数据字段。

4.按单独数据字段的形式打印每一行。

```
import struct
import string
datafile = 'ch02-fixed-width-1M.data'
# this is where we define how to
# understand line of data from the file
mask='9s14s5s'
with open(datafile, 'r') as f:
    for line in f:
        fields = struct.Struct(mask).unpack_from(line)
        print 'fields: ', [field.strip() for field in fields]
```

2.4.3 工作原理

可以用 head、more 或者类似的 Linux shell 命令来查看文件内容，然后根据所见的数据文件格式定义掩码格式。

字符串格式用来定义要提取的数据的期望显示格式。我们用格式字

符定义数据类型。因此，如果掩码定义为 9s15s5s，我们可以读作“9 个字符宽度的字符串，跟着一个 15个字符宽度的字符串，再跟上一个5个字符宽度的字符串。”

一般来说，c定义为字符（C语言中的char类型）或者长度为1的字符串，s定义为字符串（C语言中的char[]类型），d定义为浮点数（C语言中的double类型），以此类推。在 Python 官方网站上有完整的对应表，参见 http://docs.python.org/library/struct. html#format-characters。

然后逐行读取文件内容并根据指定的格式解析（通过unpack_from方法）每一行。因为在字段前面（或者后面）可能有多余的空格，用strip()方法可以去掉每个字段的前导和后导空格。

对于解包，可以使用 struct.Struct 类的面向对象（object-oriented, OO）的方式，但也可以像下面的代码这样使用非面向对象的方式：

```
fields = struct.unpack_from(mask, line)
```

两种方式唯一的不同是使用的模式。如果想用相同的格式化掩码执行更多的操作，面向对象的方法可以不必在每次调用时声明格式。而且，它让我们有能力在将来继承struct.Struct类，为特定需求进行扩展或者提供额外的功能。

2.5 从制表符分隔的文件中读取数据

另一种常见的平坦数据文件（flat datafile）格式是制表符分隔的文件。它可能导出自Excel文件，也可能是一些定制软件的输出。

庆幸的是，通常我们可以按与CSV文件几乎相同的方式来读取这种格式的文件内容。因为Python的csv模块支持的方言能让我们用相同的原则来读取相似文件格式的变体——其中一种就是制表符分割格式。

2.5.1 准备工作

此时假定我们已经知道如何读取CSV文件。如果还不清楚，请先参见2.2“从CSV文件导入数据”一节。

2.5.2 操作步骤

我们将重用2.2“从CSV文件导入数据”一节中的代码，在这里只需要改动一下使用的方言。

```
import csv
filename = 'ch02-data.tab'
data = []
try:
    with open(filename) as f:
        reader = csv.reader(f, dialect=csv.excel_tab)
    header = reader.next()
        data = [row for row in reader]
except csv.Error as e:
```

```
        print "Error reading CSV file at line %s: %s" % (reader.line_num, e)
        sys.exit(-1)
    if header:
        print header
        print '=================='
    for datarow in data:
        print datarow
```

2.5.3 工作原理

除了实例化 csv 读对象的一行代码不同，上述代码和在“从 CSV 文件导入数据”一节中的非常相似。在那行代码中，我们指定dialect参数为excel_tab方言。

2.5.4 补充说明

基于CSV格式读取数据的方式没有办法处理有“脏数据”的情况。换言之，如果有几行不是仅以换行符结尾，而是有多余的\t（制表符）标记，这时就需要在切分前对特殊行的数据进行单独清理。ch02-data-dirty.tab是含有“脏数据”的制表符分隔的文件，下面的示例代码在读取文件数据时对“脏数据”进行了清理：

```
datafile = 'ch02-data-dirty.tab'
with open(datafile, 'r') as f:
    for line in f:
        # remove next comment to see line before cleanup
        # print 'DIRTY: ', line.split('\t')
        # we remove any space in line start or end
        line = line.strip()
```

```
# now we split the line by tab delimiter
print line.split('\t')
```

我们看到了另一种分隔字段的方式——使用 split('\t')方法 。

与使用csv模块的方式相比，split()方法的优势有时候体现在：仅仅通过改变方言就可以重用相同的代码来读取数据。至于如何检测方言，可以根据文件扩展名（ .csv和.tab）或者其他一些方法（比如使用csv.Sniffer类）来判断。

2.6 从JSON数据源导入数据

本节将展示如何读取 JSON 格式的数据。此外，我们将会使用一个远程数据源。这会让本节的内容有点复杂，但同时也会使其更加实用，因为在现实世界中，我们会更多地遇到远程数据源，而不是本地数据。

JavaScript Object Notation（JSON）作为一种平台无关的格式被广泛地应用于系统间或者应用间的数据交换。

本文中，资源是我们可以读取的任何东西，可以是一个文件或者一个URL端点（可以是远程进程/程序的输出，或者一个远程静态文件）。简言之，我们不关心谁产生了数据源以及是怎么产生的，我们只需要它是一种已知的格式，如JSON。

2.6.1 准备工作

开始之前，需要安装 requests 模块，并确保可以导入到我们的虚拟环境中（在PYTHONPATH中）。在第1章“准备工作环境”中，我们已经安装了这个模块。

我们还需要能够连接网络来读取一个远程数据源。

2.6.2 操作步骤

在下述示例代码中，我们读取并解析GitHub（http://github.com）网站的最近活动时间表，操作步骤如下。

1.指定 GitHub URL 来读取 JSON 格式数据。

2.使用requests模块访问指定的URL，并获取内容。

3.读取内容并将之转化为JSON格式的对象。

4.迭代访问JSON对象，对于其中的每一项，读取每个代码库的URL值。

```
import requests
url = 'https://github.com/timeline.json'
r = requests.get(url)
json_obj = r.json()
repos = set()
for entry in json_obj:
    try:
        repos.add(entry['repository']['url'])
    except KeyError as e:
        print "No key %s. Skipping..." % (e)
from pprint import pprint
pprint(repos)
```

2.6.3 工作原理

首先，用 requests 模块获取远程资源。requests 模块提供了简单的API 来定义HTTP谓词，我们只需要发出get()方法调用，这非常简单明了。获取到数据和请求元数据后，把它们封装到 Response 对象，以供进一步处理。在本节，我们只对Response.json()方法感兴趣，这个方法可以读取Response.content的内容，把它解析成JSON并加载到JSON对象中。

现在有了JSON对象，接下来就可以处理数据了。在开始之前，需要知道数据的格式。可以用自己喜欢的浏览器或者命令行工具如wget或curl打开JSON数据源来一探究竟。

另一种方式是在IPython中获取数据，并以交互的方式查看输出。在

IPython中用命令%run program_name.py 运行程序。执行完毕后会得到程序生成的所有变量，可以使用%who或者%whos把它们列出来。

通过上述方法，我们了解了JSON数据的结构，并能够看到哪些是我们感兴趣的部分。

JSON对象基本上就是一个Python字典（或者说得更复杂些，字典的字典），我们能用大家熟知的基于 key 的符号来访问其中的一部分。通过 entry['repository'] ['url']就得到了最近更新的库中的URL列表。

通过entry['repository'['ur']可以得到实际JSON文件中的这段数据内容。

```
...
  "repository" : {
    ...
    "url" : "https://github.com/ipython/ipython",
    ...
  },
...
```

现在，我们了解了在Python代码中嵌套结构是怎样和多维key索引对应的。

2.6.4 补充说明

JSON 格式（遵循 RFC 4627 规定，参见 http://tools.ietf.org/html/rfc4627. html）最近变得非常流行，因为它比 XML 更易读而且更简洁。因此，在传输数据所需的语法上也更轻量。因为 JSON 来自 JavaScript——当今大多数富互联网应用使用的语言，使得它在 Web应用领域相当受欢迎。

Python 的 JSON 模块的功能远不止我们演示的这些，例如我们可以

特化基本的JSONEncoder/JSONDecoder类来把Python代码转换成JSON格式。经典的例子是用这种方法将Python内置的复杂数据类型变成JSON格式。

如果是简单的定制化，就不必派生JSONDecoder/JSONEncoder类，因为通过设置参数就可以解决这个问题。

例如，json.loads()会把浮点数解析成Python的float类型，在大多数情况下这都是没有问题的。不过有时候，如果JSON文件中的浮点值代表了价格，最好还是表示成十进制。我们可以告诉json解析器把浮点数转为十进制。例如，有这样一个JSON字符串。

```
jstring = '{"name":"prod1","price":12.50}'
```

接着是下面两行代码。

```
from decimal import Decimal
json.loads(jstring, parse_float=Decimal)
```

上面两行代码的输出如下。

```
{u'name': u'prod1', u'price': Decimal('12.50')}
```

2.7 导出数据到JSON、CSV和Excel

然而，在做数据可视化时，我们通常只是使用其他人的数据，所以导入和读取数据是主要工作。然而，不管是我们还是他人的需要，不管是现在还是将来的需要，确实需要把产生或者处理过的数据导出或写到某个地方。

接下来，我们将演示如何使用前面提到的Python模块导入、导出和写数据到JSON、CSV和XLSX等各种格式。

为了演示的需要，我们将使用“从定宽数据文件导入数据”一节预先生成的数据集合。

2.7.1 准备工作

对于Excel的写操作部分，需要（在虚拟环境中）安装xlwt模块。请执行下面的命令：

```
$ pip install xlwt
```

2.7.2 操作步骤

下面将介绍一段示例代码，它包括了要演示的所有格式：CSV、JSON 和 XLSX。程序的主要部分接收输入并调用合适的方法对数据进行转化。我们会逐一介绍每个代码段，并解释它们的目的。

1.导入需要的模块。

```
import os
import sys
import argparse
```

```
try:
    import cStringIO as StringIO
except:
    import StringIO
import struct
import json
import csv
```

2.然后，定义合适的读写数据的方法。

```
def import_data(import_file):
    '''
    Imports data from import_file.
    Expects to find fixed width row
    Sample row: 161322597 0386544351896 0042
    '''
    mask = '9s14s5s'
    data = []
    with open(import_file, 'r') as f:
        for line in f:
            # unpack line to tuple
            fields = struct.Struct(mask).unpack_from(line)
            # strip any whitespace for each field
            # pack everything in a list and add to full dataset
            data.append(list([f.strip() for f in fields]))
    return data
def write_data(data, export_format):
    '''Dispatches call to a specific transformer and returns data set.
    Exception is xlsx where we have to save data in a file.
```

```
    '''
    if export_format == 'csv':
        return write_csv(data)
    elif export_format == 'json':
        return write_json(data)
    elif export_format == 'xlsx':
        return write_xlsx(data)
    else:
        raise Exception("Illegal format defined")
```

3.为每一种数据格式（CSV、JSON和XLSX）分别指定各自的实现方法。

```
def write_csv(data):
    '''Transforms data into csv. Returns csv as string.
    '''
    # Using this to simulate file IO,
    # as csv can only write to files.
    f = StringIO.StringIO()
    writer = csv.writer(f)
    for row in data:
        writer.writerow(row)
    # Get the content of the file-like object
    return f.getvalue()
def write_json(data):
    '''Transforms data into json.Very straightforward.
    '''
    j = json.dumps(data)
    return j
```

```
def write_xlsx(data):
    '''Writes data into xlsx file.
    '''
    from xlwt import Workbook
    book = Workbook()
    sheet1 = book.add_sheet("Sheet 1")
    row = 0
    for line in data:
        col = 0
        for datum in line:
            print datum
            sheet1.write(row, col, datum)
            col += 1
        row += 1
        # We have hard limit here of 65535 rows
        # that we are able to save in spreadsheet.
        if row > 65535:
            print >> sys.stderr, "Hit limit of # of rows in one sheet (65535)."
            break
    # XLS is special case where we have to
    # save the file and just return 0
    f = StringIO.StringIO()
    book.save(f)
    return f.getvalue()
```

4.最后，完成main入口点代码，解析命令行参数中传入的文件路径，导入数据并导出成要求的格式。

```
if __name__ == '__main__':
```

```
# parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument("import_file", help="Path to a fixed-width data file.")
parser.add_argument("export_format", help="Export format: json, csv, xlsx.")
args = parser.parse_args()
if args.import_file is None:
    print >> sys.stderr, "You myst specify path to import from."
    sys.exit(1)
if args.export_format not in ('csv','json','xlsx'):
    print >> sys.stderr, "You must provide valid export file format."
    sys.exit(1)
# verify given path is accesible file
if not os.path.isfile(args.import_file):
    print >> sys.stderr, "Given path is not a file:%s"% args.import_file
    sys.exit(1)
# read from formated fixed-width file
data = import_data(args.import_file)
# export data to specified format
# to make this Unix-lixe pipe-able
# we just print to stdout
print write_data(data, args.export_format)
```

2.7.3 工作原理

概括地讲，首先导入定宽数据集合（在2.4“从定宽数据文件导入数

据”一节已定义），接着导出到stdout，然后可以把它存到文件，或者作为另一个程序的输入。

首先，从命令行执行程序，给定两个必选参数：输入文件名和导出文件格式（JSON、CSV和XLSX）。

成功解析这些参数后，把输入文件分派给 import_data()方法。然后，该方法返回Python数据结构（列表的列表），我们就可以方便地对其进行操作并得到合适的输出格式了。

在write_data()方法中，我们只是把请求路由给合适的方法（比如write_csv()方法）。

在 CSV 中，我们得到一个 csv.writer()实例，然后把迭代过的每一行数据写到里面。

因为将来要把输出从我们的程序重定向到另一个程序（或者仅仅是对文件执行cat操作），所以只是简单返回给定的字符串。

json模块提供的dump()方法可以很轻松地读取Python的数据结构，所以在这个例子中JSON的导出操作并不需要演示。至于CSV，我们只是简单地返回结果并把其输出给stdout。

Excel导出需要比较多的代码，因为需要创建一个更加复杂的Excel工作簿和工作单的模型来存放数据。接下来的工作和前面迭代方式相似，有两个循环，外部的循环遍历数据源集合的每一行，内部的循环遍历给定行的每一个字段。

最后，把 Book 实例保存成类文件流，这样就可以把它返回给stdout。然后，既可以把内容读取到文件中，也可以让 Web service 来消费它。

2.7.4 补充说明

当然，这仅仅是能导出的数据格式的一个小小的集合。如果想支持

更多的格式，改动起来也是相当简单的。基本上需要改动两个地方：导入和导出方法。如果想导入一种新的数据源，就需要改动导入方法。

如果想添加一种新的导出格式，首先需要添加方法来返回一个格式化了的数据流。然后，更新write_data()方法，添加新的elif分支来让它调用新的write_*方法。

另一件能做的事情就是把上述代码打成一个Python包，这样就可以在更多项目上重用它了。如果那样做的话，我们可以让数据的导入更灵活些，或者为导入添加更多的配置功能。

2.8 从数据库导入数据

通常情况是，数据分析和可视化工作处在数据管道的消费端。我们更经常使用已经生成的数据，而不是自己生成数据。例如，一个现代应用程序在关系数据库（或其他数据库）中存储了不同的数据集合，我们可以使用这些数据来生成漂亮的图表。

本节将展示在 Python 中如何使用 SQL drivers 访问数据。

本节的示例采用SQLite数据库，因为它设置起来需要的工作量最少，同时和大多数其他基于SQL的数据库引擎（MySQL和PostgreSQL）的接口相似。不过，各种数据库引擎支持的SQL方言多少有些不同。这个例子使用简单的SQL语言，因此在大多数常用的SQL数据库引擎上应该都是可以重用的。

2.8.1 准备工作

在继续本节下面的内容之前，首先需要安装SQLite库。

```
$ sudo apt-get install sqlite3
```

Python默认支持SQLite，因此不需要再安装任何与Python相关的东西。可以在IPython中执行下述代码来验证一下是否都已经安装好。

```
import sqlite3
sqlite3.version
sqlite3.sqlite_version
```

我们会得到类似下面的输出。

```
In [1]: import sqlite3
In [2]: sqlite3.version
```

Out[2]: '2.6.0'

In [3]: sqlite3.sqlite_version

Out[3]: '3.6.22'

这里，sqlite3.version返回Python的sqlite3模块的版本号，sqlite_version返回系统SQLite库的版本。

2.8.2 操作步骤

为了能够从数据库读取数据，需要以下步骤。

1.连接数据库引擎（或者是SQLite文件）。

2.在选择的表上执行查询操作。

3.读取从数据库引擎返回的结果。

本书不会讲怎样使用SQL，因为有很多专门关于这个话题的书。但为了能让大家明白，我们会解释下这个代码例子中的SQL查询语句。

SELECT ID, Name, Population FROM City ORDER BY Population DESC LIMIT 1000

这条语句从City表中查询了ID、Name和Population等列（字段）的值。ORDER BY告诉数据库引擎按照Population列对数据进行排序，同时DESC指定按降序排列。LIMIT仅允许我们获取查找到的数据的前1000条。

这个例子中，我们将使用 world.sql 示例中的表。这个表包含了全世界的城市名和人口，有超过5000条的数据。

这个表如图2-1所示。

ID	Name	Population
1024	Mumbai (Bombay)	10500000
2331	Seoul	9981619
206	São Paulo	9968485
1890	Shanghai	9696300
939	Jakarta	9604900
2822	Karachi	9269265
3357	Istanbul	8787958
2515	Ciudad de México	8591309
3580	Moscow	8389200
3793	New York	8008278
1532	Tokyo	7980230
1891	Peking	7472000
456	London	7285000
1025	Delhi	7206704
608	Cairo	6789479
1380	Teheran	6758845
2890	Lima	6464693
1892	Chongqing	6351600
3320	Bangkok	6320174
2257	Santafé de Bogotá	6260862

图2-1

首先需要把这个SQL文件导入到SQLite数据库中，代码如下。

```
import sqlite3
import sys
if len(sys.argv) < 2:
    print "Error: You must supply at least SQL script."
    print "Usage: %s table.db ./sql-dump.sql" % (sys.argv[0])
    sys.exit(1)
script_path = sys.argv[1]
if len(sys.argv) == 3:
```

```
    db = sys.argv[2]
else:
    # if DB is not defined
    # create memory database
    db = ":memory:"
try:
    con = sqlite3.connect(db)
    with con:
        cur = con.cursor()
        with open(script_path,'rb') as f:
            cur.executescript(f.read())
except sqlite3.Error as err:
    print "Error occured: %s" % err
```

这段代码会读取 SQL 文件中的 SQL 语句，然后在打开的 SQLite db 文件上执行。如果不指定db文件名，SQLite会在内存中创建一个数据库，然后逐条执行语句。

如果遇到了错误，程序会捕获异常并把错误信息打印给用户。

在把数据导入到数据库之后，就能查询数据并进行一些操作了。以下是从数据库文件读取数据的代码。

```
import sqlite3
import sys
if len(sys.argv) != 2:
    print "Please specify database file."
    sys.exit(1)
db = sys.argv[1]
try:
    con = sqlite3.connect(db)
```

```
    with con:
        cur = con.cursor()
        query = 'SELECT ID, Name, Population FROM City ORDER BY
Population DESC LIMIT 1000'
        con.text_factory = str
        cur.execute(query)
        resultset = cur.fetchall()
        # extract column names
        col_names = [cn[0] for cn in cur.description]
        print "%10s %30s %10s" % tuple(col_names)
        print "="*(10+1+30+1+10)
        for row in resultset:
            print "%10s %30s %10s" % row
except sqlite3.Error as err:
    print "[ERROR]:", err
```

2.8.3 工作原理

首先，检查用户是否提供了数据库文件路径。这只是一个快速的检查，确保我们能执行剩下的代码。

接下来尝试连接数据库。如果失败了，程序捕获到sqlite3.Error并把它打印给用户。

如果连接成功，我们通过con.cursor()得到一个游标。游标与迭代器类似，能让我们遍历数据库返回的结果集中的记录。

我们定义了一个查询操作，与数据库建立连接后，执行查询请求并通过cur.fetchall()得到结果集。如果只想获取一条结果，可以用fetchone()。

在cur.description上执行列表解析操作来得到数据库的列名。description是一个只读属性，包含了很多的信息。对每一列的信息都有一个7个元素的元组，这里只用到列名，所以仅获得每个元组的第一个元素。

接着使用简单的字符串格式化打印出带列名的表头信息，然后迭代结果集并按照类似的方式打印出每一行。

2.8.4 补充说明

数据库是当今最常见的数据源。在本小节的介绍中，我们没办法面面俱到，但建议你看看下面这些内容。

如果想查找数据库操作方面的知识，官方 Python 文档是首选。最常见的数据库是开源数据库，如MySQL、PostgreSQL和SQLite。数据库领域的另一部分是企业数据库系统，如MS SQL、Oracle 和 Sybase。Python 支持大部分的数据库，而且有抽象的接口。所以如果数据库变了，不需要改动你的程序。但可能需要一些小改动，这取决于程序是否使用了特定数据库系统的特性。例如，Oracle支持一种专门的语言PL/SQL，它不是标准的SQL。如果把数据库从 Oracle变成 MS SQL，有些地方就不工作了。类似的，SQLite 不支持 MySQL数据类型或者数据库引擎类型（MyISAM 和 InnoDB）的特性。这些事有些烦人，但让代码遵循标准SQL（http://en.wikipedia.org/wiki/SQL:2011）会让其具备数据库系统间的可移植性。

2.9 清理异常值

本节描述如何处理来自真实世界的数据集合，并介绍在做可视化前如何对数据进行清理。

我们会演示一些不同的技巧，但是它们有一个共同的目的，就是清理数据。

然而，清理的工作不应该全部被自动化。因为在应用任何健壮的现代算法来清理数据之前，我们需要了解给定的数据，需要知道异常值[1]（outlier）是什么，并且要明白展示什么数据。但是，这些内容在一节的内容中没有办法都讲清楚，因为它依赖很多方面，如统计学、领域知识和一双慧眼（然后是一点运气）。

2.9.1 准备工作

我们将使用已经熟悉的Python标准模块，不需要额外安装软件。

在本节中，我们将介绍一个新名词——MAD。在统计学上，中位数绝对偏差（Median absolute deviation，MAD）是用来描述单变量（包含一个变量）样本在定量数据中可变性的一种标准。它常用来度量统计分布，因为它会落在一组稳健统计数据中，因此对异常值有抵抗能力。

2.9.2 操作步骤

下例展示了如何用MAD来检测数据中的异常值。下面是操作步骤。

1.生成 0～1 之间的随机数据（normally distributed random data）。

2.加入一些异常值。

3.用is_outlier()方法检测异常值。

4.绘制出两个数据集合（x和filtered）的图表，观察它们的区别。

```
import numpy as np
import matplotlib.pyplot as plt
def is_outlier(points, threshold=3.5):
    """
    Returns a boolean array with True if points are outliers and False otherwise.
    Data points with a modified z-score greater than this
    # value will be classified as outliers.
    """
    # transform into vector
    if len(points.shape) == 1:
        points = points[:,None]
    # compute median value
    median = np.median(points, axis=0)
    # compute diff sums along the axis
    diff = np.sum((points - median)**2, axis=-1)
    diff = np.sqrt(diff)
    # compute MAD
    med_abs_deviation = np.median(diff)
    # compute modified Z-score
    # http://www.itl.nist.gov/div898/handbook/eda/section4/eda43.htm#
    # Iglewicz
    modified_z_score = 0.6745 * diff / med_abs_deviation
    # return a mask for each outlier
    return modified_z_score > threshold
```

```
# Random data
x = np.random.random(100)
# histogram buckets
buckets = 50
# Add in a few outliers
x = np.r_[x, -49, 95, 100, -100]
# Keep valid data points
# Note here that
# "～" is logical NOT on boolean numpy arrays
filtered = x[～is_outlier(x)]
# plot histograms
plt.figure()
plt.subplot(211)
plt.hist(x, buckets)
plt.xlabel('Raw')
plt.subplot(212)
plt.hist(filtered, buckets)
plt.xlabel('Cleaned')
plt.show()
```

注意，在NumPy中，“～”操作符被重载为一个逻辑操作符，作用在布尔数组上时为取非操作。举个例子，在pylab模式下启动IPython。

```
$ ipython –pylab
```

得到结果如下。

```
In [1]: ～numpy.array(False)
Out[1]: True
```

如图2-2所示有两个不同的直方图，第一幅图除了一个最大的异常值之外什么都没有，第二幅图中因为剔除掉了异常值，显示了多样化的

数据。

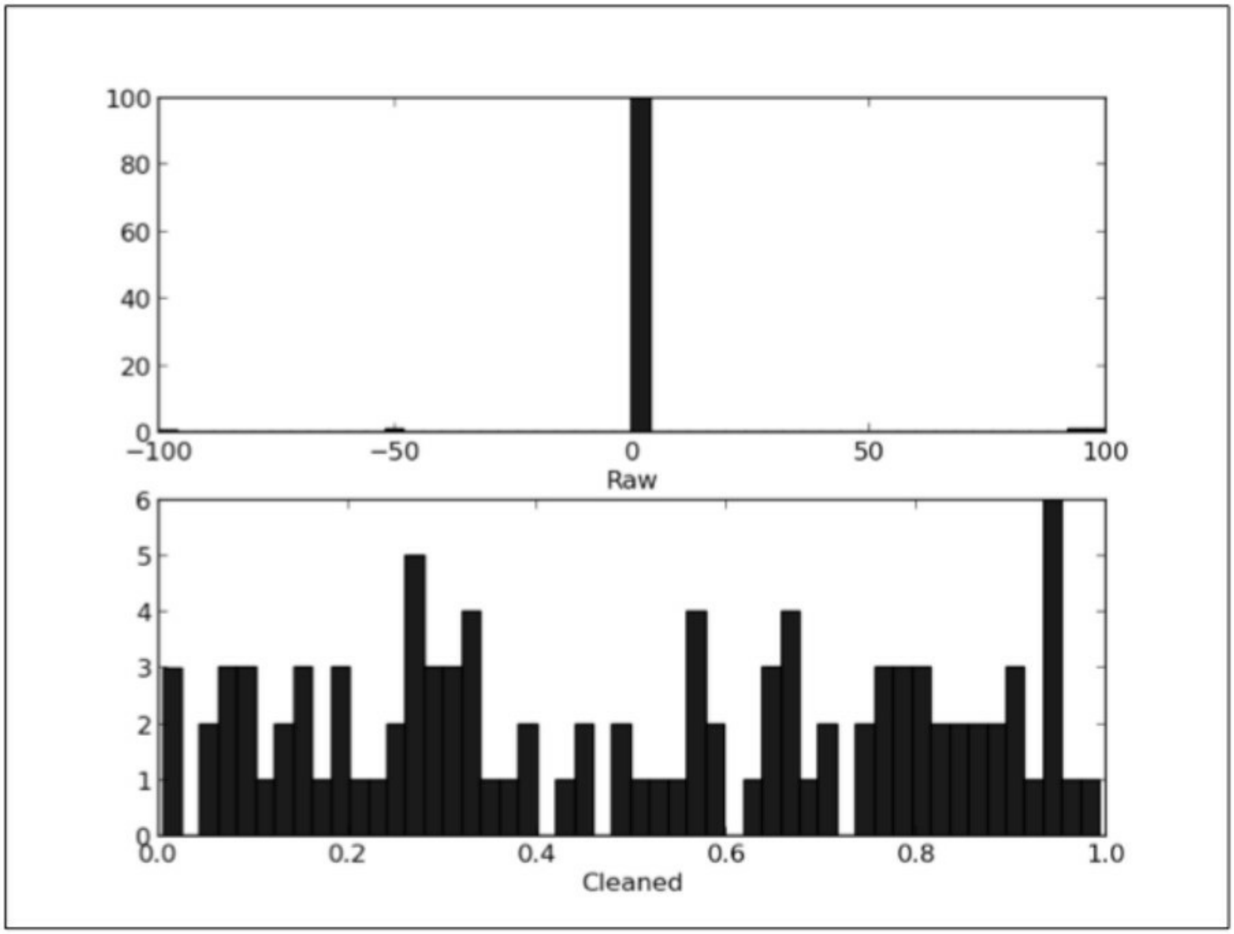


图2-2

另一种识别异常值的方法是通过人眼检查数据。我们可以创建散点图，这样能轻易地看到偏离簇中心的值，也可以绘制一个箱线图（box plot）。这样就将显示出中值、上四分位数和下四分位数，以及远离箱体的异常值点。

箱体从数据的低四分位数延伸到高四分位数，在中值处有一条线。箱体延伸出的箱须（whiskers）显示了数据的范围。超出箱须末端的点就是异常值。

下面是一段示例代码。

```
from pylab import *
# fake up some data
```

```
spread= rand(50) * 100
center = ones(25) * 50
# generate some outliers high and low
flier_high = rand(10) * 100 + 100
flier_low = rand(10) * -100
# merge generated data set
data = concatenate((spread, center, flier_high, flier_low), 0)
subplot(311)
# basic plot
# 'gx' defining the outlier plotting properties
boxplot(data, 0, 'gx')
# compare this with similar scatter plot
subplot(312)
spread_1 = concatenate((spread, flier_high, flier_low), 0)
center_1 = ones(70) * 25
scatter(center_1, spread_1)
xlim([0, 50])
# and with another that is more appropriate for
# scatter plot
subplot(313)
center_2 = rand(70) * 50
scatter(center_2, spread_1)
xlim([0, 50])
show()
```

如图2-3所示，看到由x形状标记标示出的异常值。

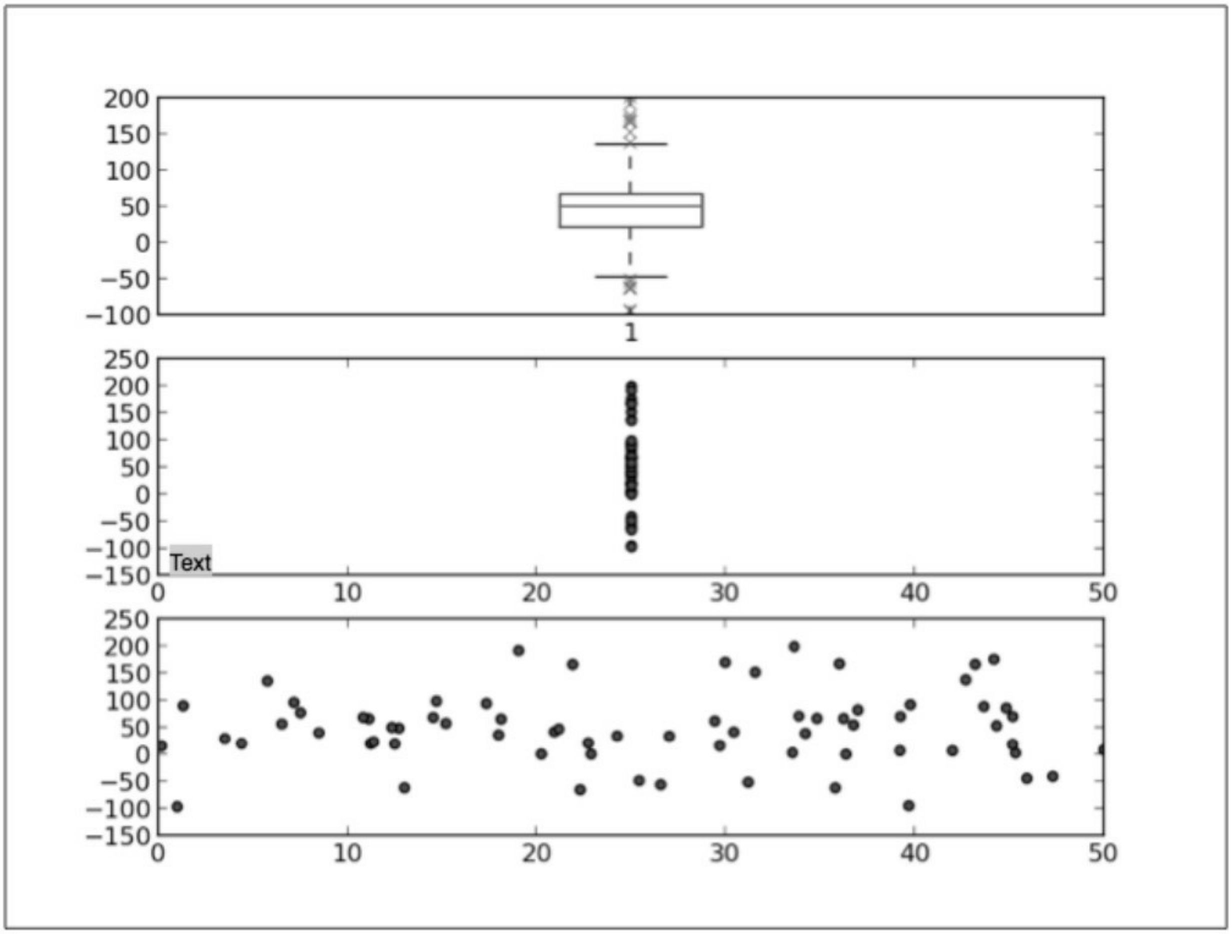


图2-3

第二幅图以散点图的形式显示了相似的数据集合。因为数据的X轴坐标值都是25，所以看起来不是很直观。另外我们无法在图上区分出负向异常值（ inlier ）和正向异常值（outlier）。

在第三幅图中，在X轴上生成的值分布在0～50的范围内，能更容易看出值与值间的不同，也能够在Y轴上看出哪些值是异常值。

在下面的代码示例中，我们将看到相同的数据（在本例中是均匀分布的）在不同的情况下看起来会截然不同，甚至有时欺骗性地传递了一些错误的信息。

```
# generate uniform data points
x = 1e6*rand(1000)
y = rand(1000)
```

```
figure()
# create first subplot
subplot(211)
# make scatter plot
scatter(x, y)
# limit x axis
xlim(1e-6, 1e6)
# create second subplot
subplot(212)
# make scatter plot
scatter(x,y)
# but make x axis logarithmic
xscale('log')
# set same x axis limit
xlim(1e-6, 1e6)
show()
```

如图2-4所示是输出的结果。

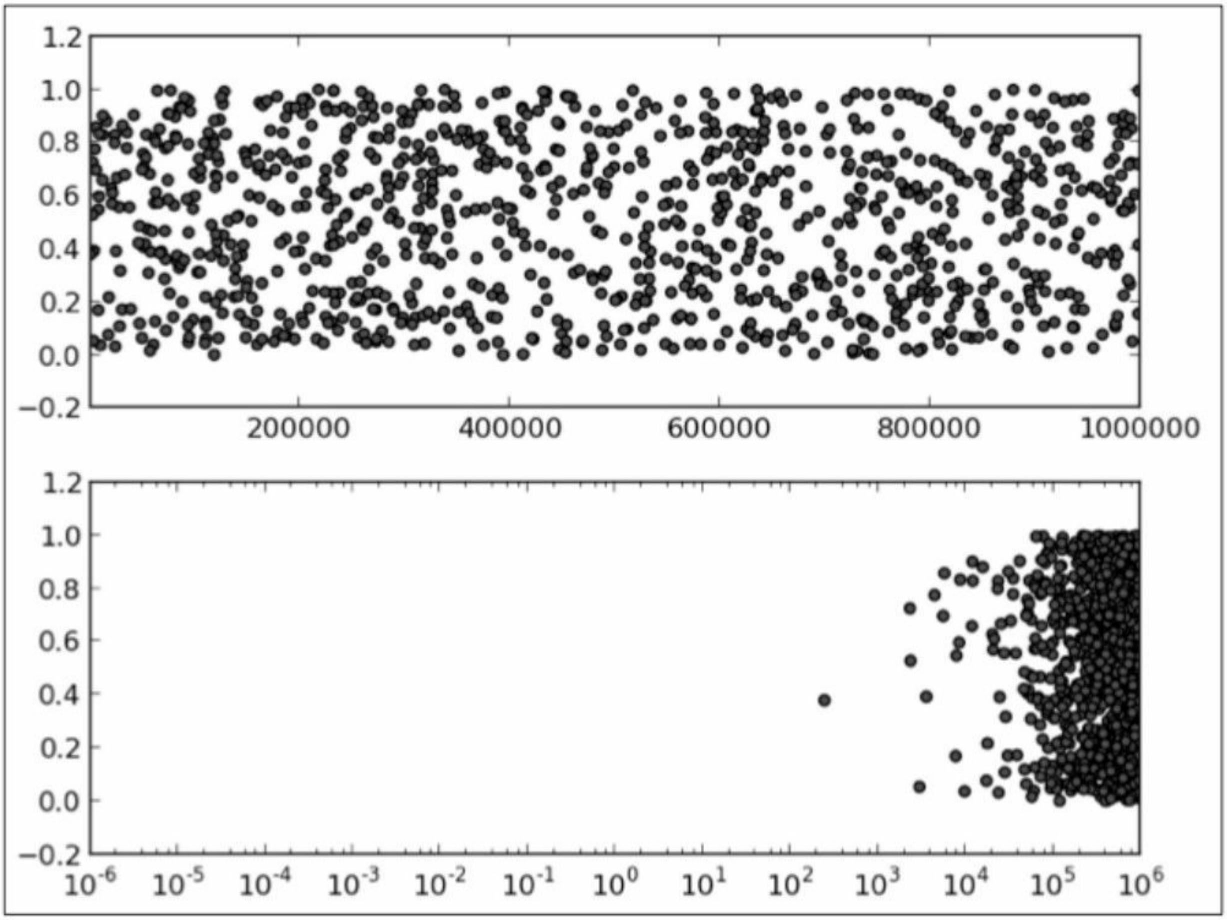


图2-4

如果数据集合有缺失值（missing value）怎么办？可以用 NumPy 加载器来补偿缺失值，或者可以写代码来把一些值替换成我们需要的值，以供进一步操作。

假如我们想把数据集合标记在一张美国地图上，在数据集合中可能有一些州名是不一致的。例如，OH、Ohio、OHIO、US-OH和OH-USA都代表美国的Ohio州。在这种情况下，我们必须把它们加载到 Microsoft Excel 或者 OpenOffice.org Calc 中，对数据集合进行手动检查。有时非常简单，只需要用Python把所有行打印出来就可以看出来。如果文件是CSV文件或者类CSV文件，可以用任何一种文本编辑器打开它，并直接检查里面的数据。

在清楚了数据的内容之后，可以写Python代码来对相似的值进行分

组，并用统一的值进行替换，以保证将来数据处理的一致性。通常的做法是，用 readlines()方法读取文件的所有行，并用标准Python字符串操作方法进行替换操作。

2.9.3 补充说明

有一些商业的和非商业的产品（如OpenRefine，参见https://github.com/ OpenRefine），提供了针对实时“脏”数据集合的一些自动化处理服务。

即便这样，清理异常值的过程还是需要人工参与的。人工参与的多少取决于数据的噪声程度和对数据的理解程度。

如果想学习更多关于异常值清理和常规数据清理的知识，可以看一下概率模型（statistical models）和采样理论（sampling theory）。

2.10 读取大块数据文件

Python非常擅长处理文件或类文件对象的读写。例如，如果你想加载一个几百MB的大文件，假如你有一个至少 2GB 内存的现代计算机，Python 处理起来也不会有任何问题。因为它不会一次性地加载所有内容，而是聪明地按照需要来加载。

即使对于相当大的文件，做一些像下面代码这样简单的操作也是很轻松的。

```
with open('/tmp/my_big_file', 'r') as bigfile:
    for line in bigfile:
        # line based operation, like 'print line'
```

但是如果想跳到文件中的某一个特定位置，或者执行一些非顺序的读操作，我们需要手工写代码来调用一些对大多数用户都足够灵活的IO方法，如seek()、tell()、read()和next()。这些方法大多数仅仅是绑定到了C实现上（根据操作系统特定的实现），因此运行会非常快，但是根据操作系统的不同，方法的表现会有所不同。

2.10.1 操作步骤

有时大文件的处理可以按文件块进行，这取决于我们的目的是什么。例如，可以读取1000行，然后用Python标准的基于迭代器的方法进行处理，代码如下。

```
import sys
filename = sys.argv[1] # must pass valid file name
with open(filename, 'rb') as hugefile:
```

```
chunksize = 1000
readable = ''
# if you want to stop after certain number of blocks
# put condition in the while
while hugefile:
    # if you want to start not from 1st byte
    # do a hugefile.seek(skipbytes) to skip
    # skipbytes of bytes from the file start
    start = hugefile.tell()
    print "starting at:", start
    file_block = '' # holds chunk_size of lines
    for _ in xrange(start, start + chunksize):
        line = hugefile.next()
        file_block = file_block + line
        print 'file_block', type(file_block), file_block
    readable = readable + file_block
    # tell where are we in file
    # file IO is usually buffered so tell()
    # will not be precise for every read.
    stop = hugefile.tell()
    print 'readable', type(readable), readable
    print 'reading bytes from %s to %s' % (start, stop)
    print 'read bytes total:', len(readable)
    # if you want to pause read between chucks
    # uncomment following line
    # raw_input()
```

在Python命令行解释器中调用上面的代码，给定文件名作为第一个

参数。

```
$ python ch02-chunk-read.py myhugefile.dat
```

2.10.2 工作原理

我们希望能够读取成块的文件行并进行处理，而不必把整个文件读取到内存中。

首先，打开文件，在 for 循环内部读取文件行。在文件中的移动是通过在文件对象上调用next()来完成的。这个方法读取文件中的一行，然后把文件指针移到下一行。为了简化示例代码，我们只是把file_block加到输出变量readable上，没有进行任何处理。

在执行过程中的一些打印操作是为了说明某些变量的当前状态。

while循环中的最后一行注释代码是raw_input()。如果去掉注释的话，就可以输出在这一句之前打印的文件行，并暂停程序的执行。

2.10.3 补充说明

当然，本节介绍的只是读取大文件的众多方法中的一种。其他方法可能会引入一些特定的Python库或C库，但这完全取决于我们要对数据做什么，以及如何操作数据。

并行方法如MapReduce范式最近非常流行，因为它能让我们以低成本获得更大的处理能力和内存空间。

多进程处理（multiprocessing）有时也是一个可行的方法。Python针对创建和管理线程提供了很好的库支持，如multiprocessing、threading和thread。

如果项目中会重复地处理大文件，我们建议建立自己的数据管道，这样每次需要数据以特定形式输出时，不必再找到数据源进行手动处理。

2.11 读取流数据源

如果数据是来自一个连续的数据源呢？如果需要读取连续数据呢？接下来，本节将介绍一个适用于许多真实场景的简单解决方案。然而它并不是通用的，需要针对个人应用中的特殊情况进行调整。

2.11.1 操作步骤

在本节中，我们将向你演示如何读取一个实时变化的文件，并把输出打印出来。我们将使用普通的Python模块来完成它，代码如下。

```
import time
import os
import sys
if len(sys.argv) != 2:
    print >> sys.stderr, "Please specify filename to read"
filename = sys.argv[1]
if not os.path.isfile(filename):
    print >> sys.stderr, "Given file: \"%s\" is not a file" % filename
with open(filename,'r') as f:
    # Move to the end of file
    filesize = os.stat(filename)[6]
    f.seek(filesize)
    # endlessly loop
    while True:
        where = f.tell()
```

```
# try reading a line
line = f.readline()
# if empty, go back
if not line:
    time.sleep(1)
    f.seek(where)
else:
    # , at the end prevents print to add newline, as readline()
    # already read that.
    print line,
```

2.11.2 工作原理

代码的核心部分在 while True:循环中。这个循环永远不会停止（除非在键盘上键入 Ctrl+C 来中断它）。首先，将文件指针移动到文件末尾，然后试着读取文件中的一行。如果没有读出内容，意味着在用seek()方法检查之后文件中没有添加内容。就这样，等待一秒然后重试。

如果读到了一行内容，就把它打印出来，因为文件行末尾已经有换行符，在打印时不需要再向末尾添加换行符。

2.11.3 补充说明

我们可能想读取最后的n行，这就要把文件指针移动到文件末尾前的某个地方。可以通过 file.seek(filesize – N * avg_line_len)把文件指针移到那里。这里的avg_line_len应该是近似的平均行长度（大约1024）。然后，可以用readlines()从那个点开始读文件行，然后打印出列表中的[-N]行。

本例中的概念可以用在许多解决方案上。例如，如果输入是一个类文件对象或者一个远程HTTP资源，就可以从远程服务读取输入信息，并持续地解析它，然后实时地更新图表，或者更新到中间队列（intermediate queue）、缓冲或者数据库。

io模块非常适用于流处理。Python从2.6版本开始支持它，并作为文件模块的替代品。io 模块在 Python 3.x 中已经是一个默认接口。

在一些更复杂的数据管道中，需要启用消息队列（message queue）。到达的连续数据会被放在队列里一段时间，然后才能被我们接收到。 这样做的好处是作为数据的使用者，我们有能力在数据过载时暂停处理。而且，把数据放在通用的消息总线（message bus）中，能够让我们项目中的客户去使用同样的数据，同时又不会干涉到我们的软件。

2.12 导入图像数据到NumPy数组

接下来会介绍如何用NumPy和SciPy这两种Python库来做图像处理。

在科学计算中，图像通常被看做n维数组。图像一般是二维数组，在我们的例子中，它们会被表示为NumPy数组数据结构。因此，对图像执行的一些方法及操作被看作是矩阵操作。

从矩阵操作这个意义上讲，图像不需要总是二维的。在医疗或者生物科学领域，图像是更高维度的数据结构，比如3D（有表示深度的Z轴或者时间轴）或者4D（有三个空间维度和一个时间维度）。但是在本节我们不会用到这些。

可以用各种方法导入图像，这完全取决于你想对图像做什么操作。并且，这也取决于你所使用的工具的生态系统以及项目所运行的平台。

在本节中，我们将演示Python处理图像的几种方式，它们更多的是和科学处理相关，与图像操作艺术方面的关系不大。

2.12.1 准备工作

本节的一些例子将使用SciPy库。如果你安装了NumPy，SciPy库也就已经安装好了。如果还没有，用操作系统的包管理工具也可以很方便地安装，执行下面的命令。

$ sudo apt-get install python-scipy

对于Windows用户，我们推荐用预打包的Python环境，如EPD。这在第1章“准备工作环境”已经讨论过。

如果想用官方发布的源码进行安装，请确保已经安装了相应的系统

依赖项如下所示。

- ◆ BLAS 和LAPACK：libblas 和 liblapack。
- ◆ C 和 Fortran 编译器：gcc 和 gfortran。

2.12.2 操作步骤

任何一个工作在数字信号处理领域，或者曾经参加过数字信号处理或相关学科的大学课程的人，都会遇到Lena图。Lena图实际上是一幅标准图，用来验证图像处理算法。

SciPy已经把这幅图打包在了misc模块中，因此我们可以很简单地重用这幅图。下面是获取并显示这幅图的代码。

```
import scipy.misc
import matplotlib.pyplot as plt
# load already prepared ndarray from scipy
lena = scipy.misc.lena()
# set the default colormap to gray
plt.gray()
plt.imshow(lena)
plt.colorbar()
plt.show()
```

代码会打开一个新窗口，显示Lena图的灰度图和坐标轴。颜色条显示了图像上值的范围，在这里显示的是0——黑色到255——白色（如图2-5所示）。

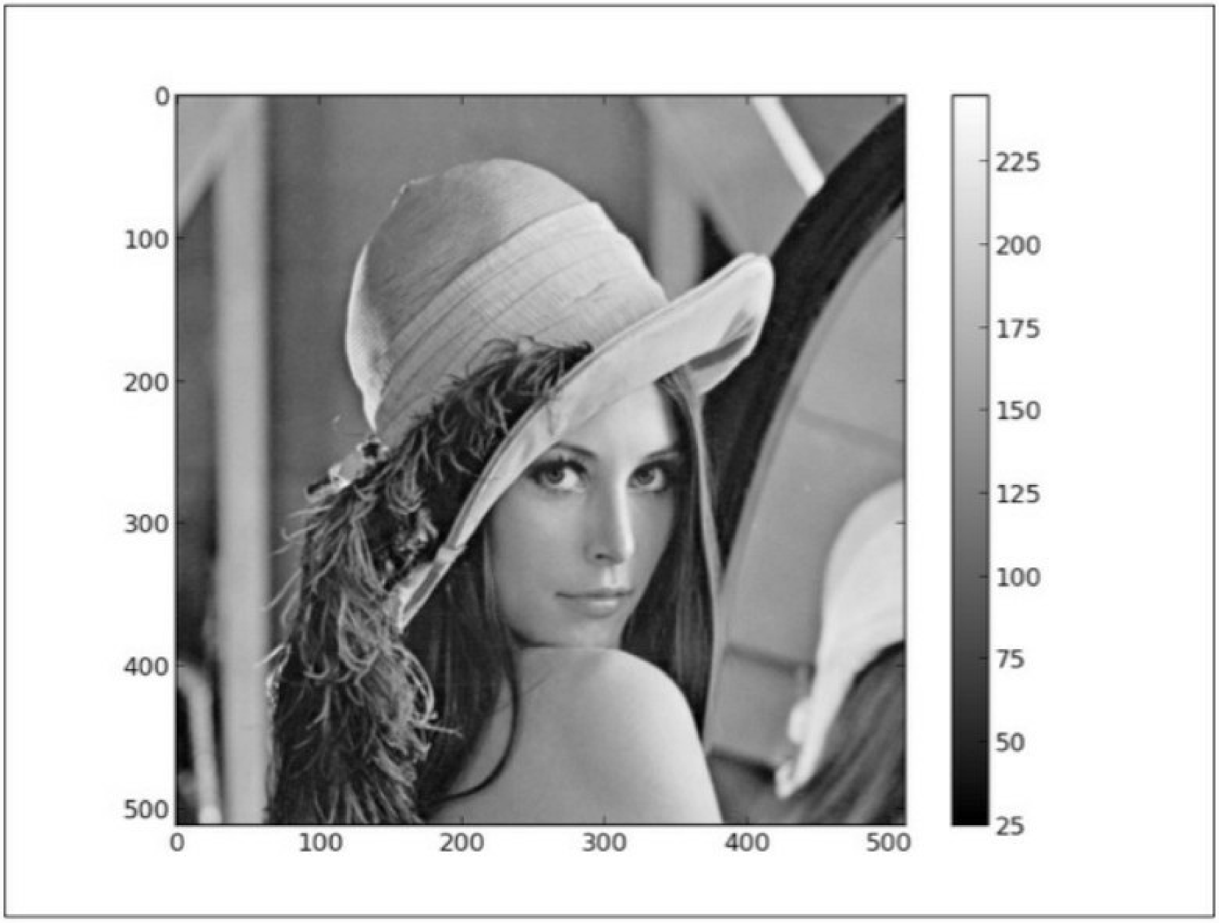


图2-5

更进一步，可以通过下面的代码来检查这个对象。

```
print lena.shape
print lena.max()
print lena.dtype
```

上面代码的输出如下：

```
(512, 512)
245
dtype('int32')
```

看到图像信息如下。

- ◆ 512 个点宽和 512 个点高。
- ◆ 整个数组（图像）的最大值是 245[2]。

◆ 每个点都被表示为小端（little endian）32 位整数。

也可以用 Python Image Library（PIL）读入图像。在第 1 章“准备工作环境”中我们已经安装好了PIL。

```
import numpy
import Image
import matplotlib.pyplot as plt
bug = Image.open('stinkbug.png')
arr = numpy.array(bug.getdata(), numpy.uint8).reshape(bug.size[1],
bug.size[0], 3)
plt.gray()
plt.imshow(arr)
plt.colorbar()
plt.show()
```

也可以用与处理Lena图相似的方式观察其他图像，如图2-6所示。

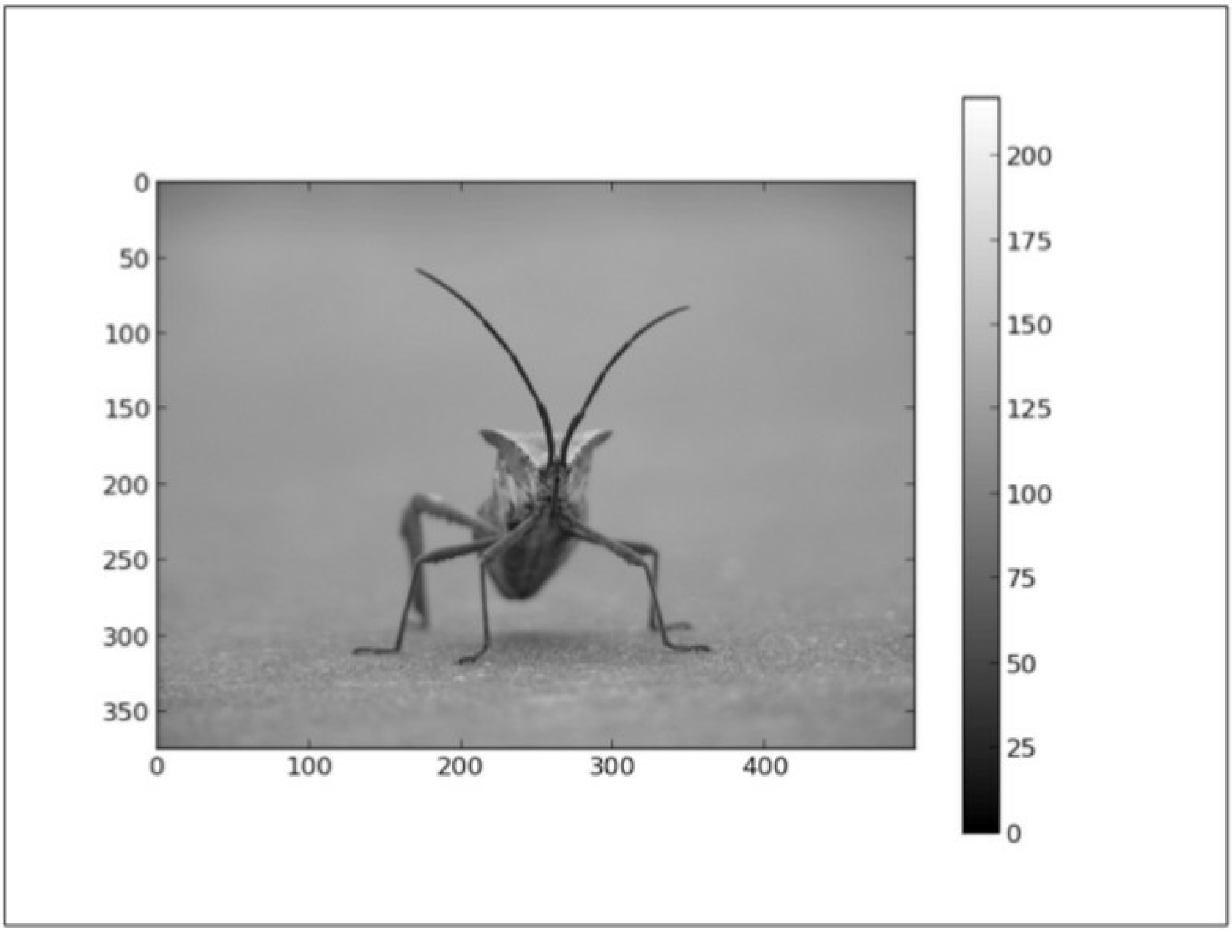


图2-6

如果我们工作在一个用PIL作为其默认的图像加载器的系统上，上面的内容或许可以帮到你。

2.12.3 工作原理

除了简单加载图像，我们真正想做的是用Python操作并处理图像。假如，我们想加载一幅包含RGB通道的真实图像，把它转换成单通道的ndarray，然后用数组切片的方法来放大部分图像。下面的代码演示了如何用NumPy和matplotlib完成这些工作。

```
import matplotlib.pyplot as plt
import scipy
```

```
import numpy
bug = scipy.misc.imread('stinkbug1.png')
# if you want to inspect the shape of the loaded image
# uncomment following line
#print bug.shape
# the original image is RGB having values for all three
# channels separately. We need to convert that to greyscale image
# by picking up just one channel.
# convert to gray
bug = bug[:,:,0]
```

bug[:,:,0]称作数组切片（array slicing）。NumPy 的这个功能让我们能够选取多维数组中任意部分。例如，让我们看如下一维数组。

```
>>> a = array(5, 1, 2, 3, 4)
>>> a[2:3]
array([2])
>>> a[:2]
array([5, 1])
>>> a[3:]
array([3, 4])
```

对多维数组，用逗号区别不同的维度。示例如下：

```
>>> b = array([[1,1,1],[2,2,2],[3,3,3]]) # matrix 3 x 3
>>> b[0,:] # pick first row
array([1,1,1])
>>> b[:,0] # we pick the first column
array([1,2,3])
```

看一下下面这段代码。

```
# show original image
```

```
plt.figure()
plt.gray()
plt.subplot(121)
plt.imshow(bug)
# show 'zoomed' region
zbug = bug[100:350,140:350]
```

上述代码放大了整图的某个一部分。请记住，图像不过是一个被表示为NumPy数组的多维数组。在这里，放大的意思是在矩阵中选择一个行和列范围。我们选择了从100行到250行，从140列到350列之间的部分矩阵。切记，数组下标从0开始，坐标上的100实际上是第101行。

```
plt.subplot(122)
plt.imshow(zbug)
plt.show()
```

结果显示如图2-7所示。

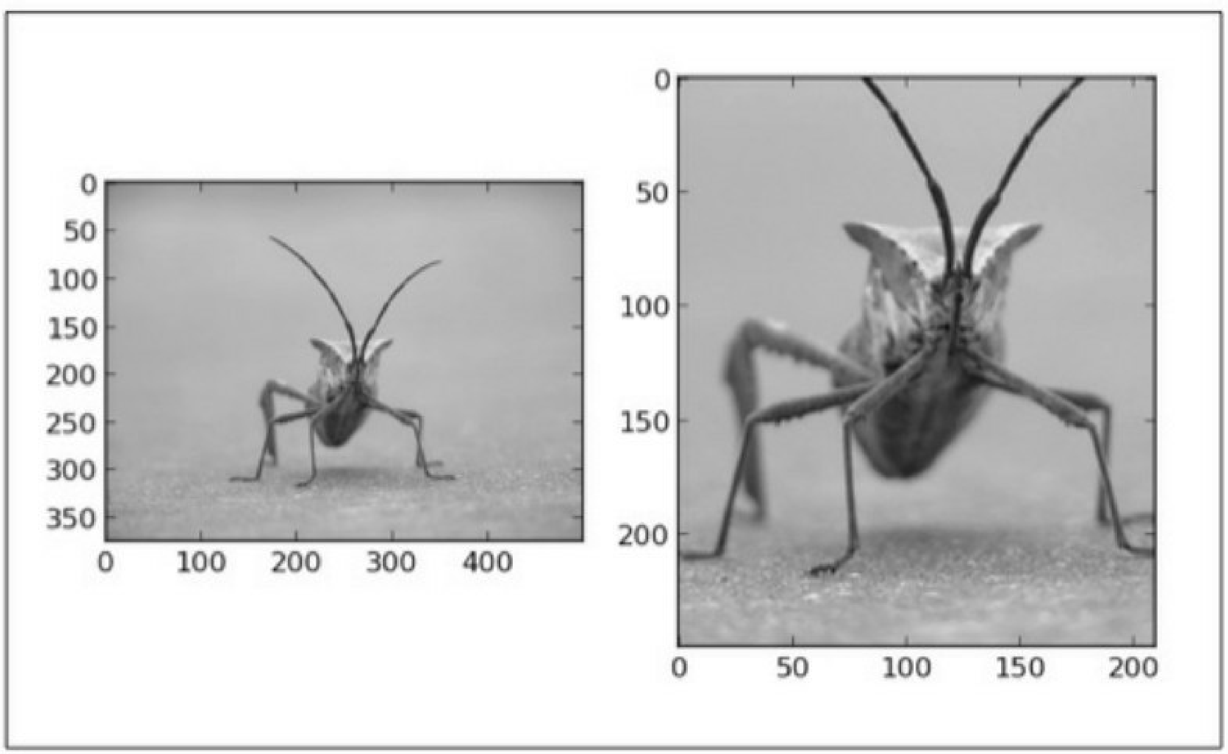


图2-7

2.12.4 补充说明

对于大图像，我们推荐使用 numpy.memmap 来做图像的内存映射，因为这会加快操作图像的速度。例如：

```
import numpy
file_name = 'stinkbug.png'
image = numpy.memmap(file_name, dtype=numpy.uint8, shape = (375, 500))
```

代码把一个大文件的一部分加载到内存中，并当作NumPy数组来访问它。这样操作的效率非常高，因为它允许我们像标准NumPy数组那样操作文件数据结构，同时又不用把所有内容全部加载到内存中。shape参数定义了数组的形状，数组由file_name参数指定的类文件对象加载。注意，Python 中的 mmap（http://docs.python.org/2/ library/mmap.html）有类似的概念，但很重要的一点区别是，NumPy的memmap返回类数组对象，而Python的mmap 返回一个类文件对象。因此它们在用法上有很大的不同，不过这些不同在它们各自的使用环境中还是很合适的。

有一些专注于图像处理的专业软件包，如 scikit-image（http://scikit-image.org/）。它们构建在NumPy/SciPy库之上，基本上是图像处理算法的免费合集。如果想做边缘检测、图像去噪，或者轮廓查找，可以从scikit工具中查找相应的算法。学习scikit最好的方法是看示例库，并找到其对应图像和代码（http://scikit-image.org/docs/dev/auto_examples/）。

2.13 生成可控的随机数据集合

本节将展示生成随机数字序列和单词序列的不同方法。一些例子使用标准Python模块，一些使用NumPy/SciPy方法。

我们会接触到一些统计学术语，但是我们会逐一解释这些术语，所以在读本节内容时你不必拿着一本统计学参考书。

我们用常用的Python模块生成一些数据集合。然后，就可以用这些数据来了解分布、方差、采样和一些类似的统计学术语。更重要的是，可以用假数据来了解统计方法是不是能够得到我们想要的模型。因为已经预先知道了模型，所以我们可以把统计方法应用到已知的数据上进行验证。在真实场景下，我们是没办法做到这一点的，因为我们必须要估计到，总会有一定程度的不确定性因素存在，可能导致错误的结果。

2.13.1 准备工作

在练习这些示例的时候，不需要安装任何新的东西。但是有一些统计学的知识是有帮助的，虽然不是必需的。

这里有一个简短的术语表可以补充一下统计学知识。在本章和接下来几章会用到这些术语。

◆ 分布或者概率分布（Distribution or probability distribution）：表示统计实验的结果和发生概率之间的联系。

◆ 标准差（Standard deviation）：这个数值表示个体和群体之间的差异。如果差异很大，标准差会比较大；如果所有个体实验在整组范围内基本相同，标准差会比较小。

◆ 方差（Variance）：标准差的平方。

◆ 总体或者统计总体（Population or statistical population）：所有潜在的可观测案例的集合。例如，如果我们对世界上学生的平均成绩感兴趣，那么统计总体就是世界上所有学生的成绩。

◆ 样本（Sample）：这是总体的子集。我们无法拿到世界上所有学生的所有成绩，因此只能收集抽样数据并对之进行建模。

2.13.2 操作步骤

可以用Python的random模块生成一个简单的随机数样本。请看下面的例子：

```
import pylab
import random
SAMPLE_SIZE = 100
# seed random generator
# if no argument provided
# uses system current time
random.seed()
# store generated random values here
real_rand_vars = []
# pick some random values
real_rand_vars = [random.random() for val in xrange(SIZE)]
# create histogram from data in 10 buckets
pylab.hist(real_rand_vars, 10)
# define x and y labels
pylab.xlabel("Number range")
pylab.ylabel("Count")
# show figure
```

pylab.show()

这是一个均匀分布的样本。当我们运行示例代码时，可以看到如图2-8所示的图。

尝试设置SAMPLE_SIZE为一个大数（如10000），观察直方图是如何变化的。

如果想让值的区间从 0～1 变为从 1～6（例如，模拟掷一个色子），可以用random.randint(min, max)。这里的 min 和 max 指相应的下限和上限。如果想生成浮点数而不是整数的样本，可以用random.uniform(min, max)方法。

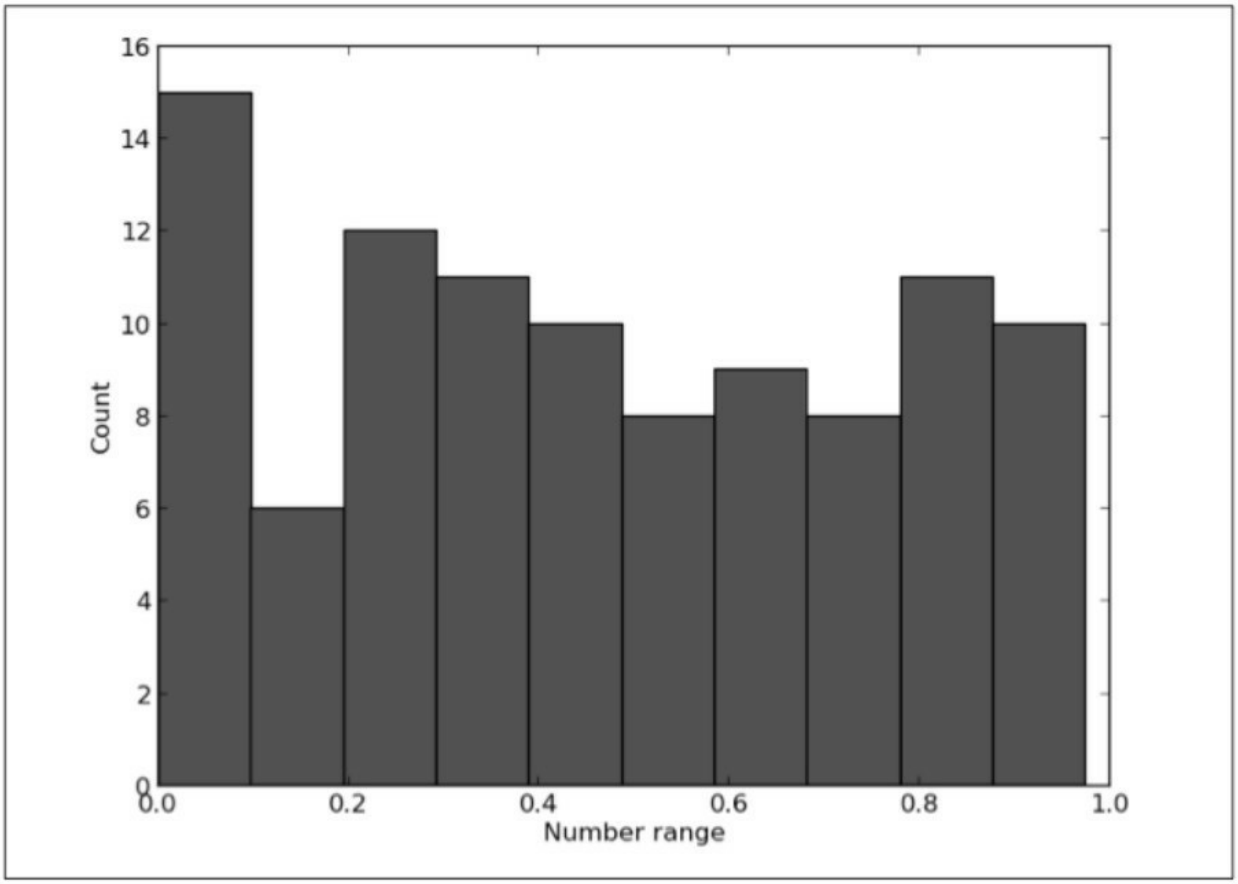


图2-8

用相似的方式，使用相同的工具，可以生成虚拟价格增长数据的时序图，并加上一些随机噪声。

import pylab

```
import random
# days to generate data for
duration = 100
# mean value
mean_inc = 0.2
# standard deviation
std_dev_inc = 1.2
# time series
x = range(duration)
y = []
price_today = 0
for i in x:
    next_delta = random.normalvariate(mean_inc, std_dev_inc)
    price_today += next_delta
    y.append(price_today)
pylab.plot(x,y)
pylab.xlabel("Time")
pylab.xlabel("Time")
pylab.ylabel("Value")
pylab.show()
```

这段代码定义了100个数据点（虚拟天数）的序列。对于接下来的每一天，从中值为mean_inc，标准差为std_dev_inc的正态分布（random.normalvariate()）中选取一个随机值，然后加上前一天的价格（price_today）作为当天的价格。

如果想要更多的控制，可以使用不同的分布。下面的代码说明并展示了不同的分布。在演示它们时，我们会注意解释每一个代码段。我们从导入需要的模块开始，然后对几个直方图进行说明。我们也创建了一

个图来容纳并显示所有的直方图。

```
# coding: utf-8
import random
import matplotlib
import matplotlib.pyplot as plt
SAMPLE_SIZE = 1000
# histogram buckets
buckets = 100
plt.figure()
# we need to update font size just for this example
matplotlib.rcParams.update({'font.size': 7})
```

为了能排列下所有规定的图形，我们定义了一个由6×2的subplot网格来显示所有的直方图。第一个图形是在[0,1)之间分布的随机变量（normal distributed random variable）。

```
plt.subplot(621)
plt.xlabel("random.random")
# Return the next random floating point number in the range [0.0, 1.0).
res = [random.random() for _ in xrange(1, SAMPLE_SIZE)]
plt.hist(res, buckets)
```

我们绘制的第二个图形是一个均匀分布的随机变量（uniformlydistributedrandomvariable）。

```
plt.subplot(622)
plt.xlabel("random.uniform")
# Return a random floating point number N such that a <= N <= b for a <= b and b <= N <= a for b < a.
# The end-point value b may or may not be included in the range depending on floating-point rounding in the equation a + (b-a) *
```

random().

a= 1

b = SAMPLE_SIZE

res = [random.uniform(a, b) for _ in xrange(1, SAMPLE_SIZE)]

plt.hist（res,buckets）

第三个图形是一个三角形分布（triangular distribution）。

plt.subplot(623)

plt.xlabel("random.triangular")

Return a random floating point number N such that low <= N <= high and with the specified

mode between those bounds. The low and high bounds default to zero and one. The mode

argument defaults to the midpoint between the bounds, giving a symmetric distribution.

low = 1

high = SAMPLE_SIZE

res = [random.triangular(low, high) for _ in xrange(1, SAMPLE_SIZE)]

plt.hist(res, buckets)

第四个图形是 beta 分布（beta distribution）。参数的条件是 alpha 和 beta 都要大于 0，返回值在0～1之间。

plt.subplot(624)

plt.xlabel("random.betavariate")

alpha = 1

beta = 10

res = [random.betavariate(alpha, beta) for _ in xrange(1, SAMPLE_SIZE)]

plt.hist(res, buckets)

第五幅图显示了一个指数分布（exponential distribution）。lambd的值是1.0除以期望的中值，是一个不为零的数（参数应该叫做lambda，但它是Python的一个保留字）。如果lambd是整数，返回值的范围是零到正无穷大；如果lambd为负，返回值范围是负无穷大到零。

```
plt.subplot(625)
plt.xlabel("random.expovariate")
lambd = 1.0 / ((SAMPLE_SIZE + 1) / 2.)
res = [random.expovariate(lambd) for _ in xrange(1, SAMPLE_SIZE)]
plt.hist(res, buckets)
```

下一幅图是 gamma 分布（gamma distribution），要求参数 alpha 和 beta 都大于零。概率分布函数如下。

$$PDF(x) = \frac{x^{a-1} e^{\frac{-x}{\beta}}}{\gamma(a)\beta^{a}}$$

下面是gamma分布的代码。

```
plt.subplot(626)
plt.xlabel("random.gammavariate")
alpha = 1
beta = 10
res = [random.gammavariate(alpha, beta) for _ in xrange(1, SAMPLE_SIZE)]
plt.hist(res, buckets)
```

下一幅图是对数正态分布（Log normal distribution）。如果取这个分布的自然对数，会得到一个中值为mu，标准差为sigma的正态分布。mu可以取任何值，sigma必须大于零。

```
plt.subplot(627)
```

```
plt.xlabel("random.lognormvariate")
mu = 1
sigma = 0.5
res = [random.lognormvariate(mu, sigma) for _ in xrange(1, SAMPLE_SIZE)]
plt.hist(res, buckets)
```

下一幅图是一个正态分布（normal distribution），中值为 mu，标准差为 sigma。

```
plt.subplot(628)
plt.xlabel("random.normalvariate")
mu = 1
sigma = 0.5
res = [random.normalvariate(mu, sigma) for _ in xrange(1, SAMPLE_SIZE)]
plt.hist(res, buckets)
```

最后一幅图是帕累托分布（Pareto distribution），alpha 是形状参数。

```
plt.subplot(629)
plt.xlabel("random.paretovariate")
alpha = 1
res = [random.paretovariate(alpha) for _ in xrange(1, SAMPLE_SIZE)]
plt.hist(res, buckets)
plt.tight_layout()
plt.show()
```

虽然这个示例代码内容有点多，但基本上讲，我们选取了1000个随机数，演示了几种不同的分布。这些都是应用在不同统计学分支中（经济学、社会学、生物科学等）的常见分布。

我们应该看到基于不同分布算法的直方图之间的区别。不妨花些时间来理解一下这 9幅图（如图2-9所示）。

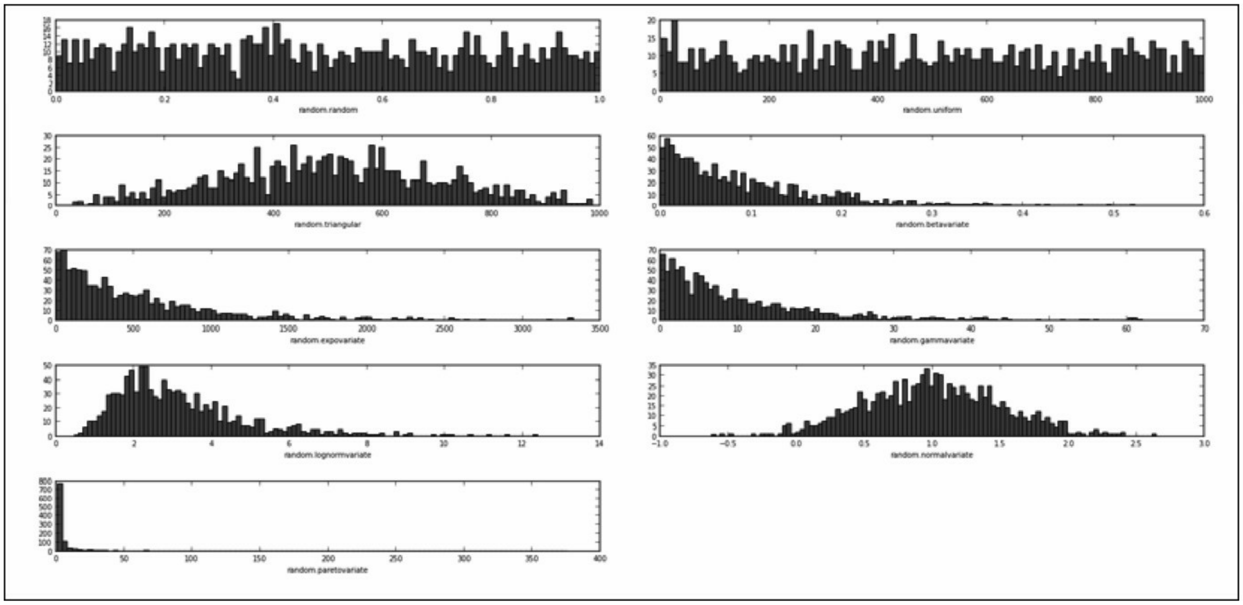


图2-9

用 seed()来初始化伪随机数生成器，这样 random()方法就能生成相同的期望随机值。有时候这非常有用，并且比预先生成随机数并保存到文件中要好。第二种方法并不总是可行的，因为它要求保存（可能是大量的）数据到文件系统。

如果想避免随机生成的序列重复，我们推荐使用random.SystemRandom，其底层使用os.urandom。os.urandom 提供了对更多熵源（entropy source）的访问。如果使用这个随机数生成器接口，seed()和setstate()没有影响。这样一来，样本就不是可重现的了。

如果想要一些随机的单词，（在 Linux 系统中）最简单的方法可能就是用/usr/share/dict/words了。从下面的例子中，我们可以看到是如何做的。[3]

```
import random
with open('/usr/share/dict/words', 'rt') as f:
    words = f.readlines()
```

```
words = [w.rstrip() for w in words]
for w in random.sample(words, 5):
    print w
```

这个方案仅仅是针对Unix系统的，在Windows上不可行（但可在Mac上运行）。Windows用户可以使用从各种免费的资源（Project Gutenberg、Wiktionary、British National Corpus或者 Dr Peter Norvig 的 http://norvig.com/big.txt）生成的文件。

2.14 真实数据的噪声平滑处理

本节将引入一些高级算法，帮助我们清理来自真实数据源的数据。这些算法在信号处理领域很有名，我们不会深究其数学上的实现，但会举例说明为什么它们是可行的，以及它们的工作原理和应用场景。

2.14.1 准备工作

来自各种真实世界传感器的数据通常是不平滑和不干净的，包含了一些我们不想显示在图表或图形中的噪声。我们希望图表和图形能清晰地传递信息，不想让用户在理解上花费过多的精力。

在这里，我们不需要安装任何新的软件，因为接下来我们将使用一些已经熟悉的Python软件包：NumPy、SciPy和matplotlib。

2.14.2 操作步骤

基础算法是基于滚动窗口（rolling window）模式（例如卷积）。窗口滚动过数据，然后计算出窗口内数据的平均值。

对于离散数据，我们使用NumPy的convolve方法，它返回两个一维序列的离散线性卷积。我们也使用NumPy的linspace方法，它生成一个给定间隔的等距数字序列。

方法ones定义了一个所有元素值为1的序列或者矩阵（例如多维数组）。我们用它来生成用于求平均值的窗口。

2.14.3 工作原理

平滑数据噪声的一个简单朴素的做法是，对窗口（样本）求平均，

然后仅仅绘制出给定窗口的平均值，而不是所有的数据点。这也是更高级算法的基础。

```
from pylab import *
from numpy import *
def moving_average(interval, window_size):
    '''Compute convoluted window for given size
    '''
    window = ones(int(window_size)) / float(window_size)
    return convolve(interval, window, 'same')
t = linspace(-4, 4, 100)
y = sin(t) + randn(len(t))*0.1
plot(t, y, "k.")
# compute moving average
y_av = moving_average(y, 10)
plot(t, y_av,"r")
#xlim(0,1000)
xlabel("Time")
ylabel("Value")
grid(True)
show()
```

如图2-10所示，可以看出平滑处理后的曲线和原始数据点（图上的点）之间的对比情况。

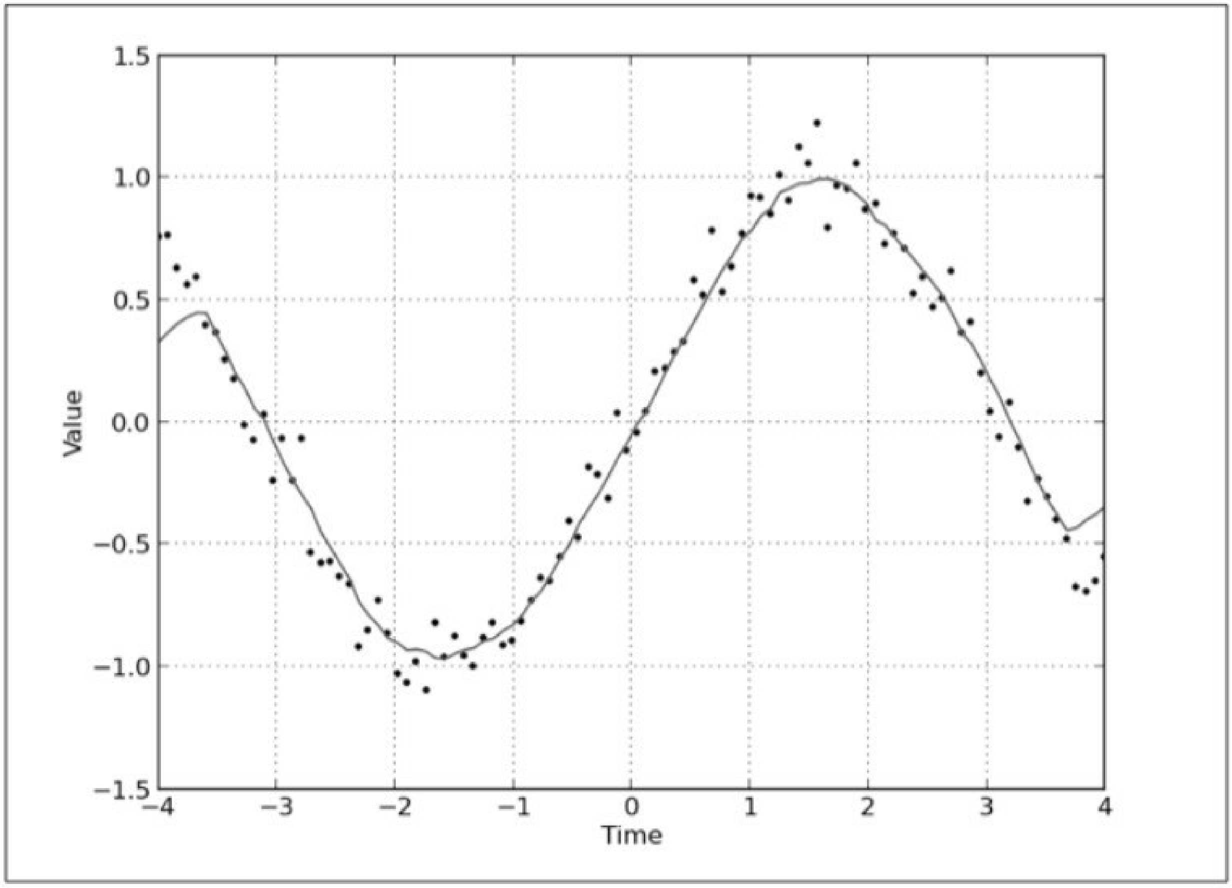


图2-10

沿着这种思路，我们可以开始一个更高级的例子。在这个例子中我们将使用现有的SciPy库来让窗口平滑处理达到更好的效果。

以下方法是基于信号（指数据点）窗口的卷积（函数的总和）。我们在准备信号时用了一些小技巧，向两端添加相同信号的副本并做反射。这样一来，我们就减小了数据的边界效应。这段代码是SciPy Cookbook一书中的例子，参见http://www.scipy.org/Cookbook/SignalSmooth。

```
import numpy
from numpy import *
from pylab import *
# possible window type
WINDOWS = ['flat', 'hanning', 'hamming', 'bartlett', 'blackman']
```

```
# if you want to see just two window type, comment previous line,
# and uncomment the following one
# WINDOWS = ['flat', 'hanning']
def smooth(x, window_len=11, window='hanning'):
    """
    Smooth the data using a window with requested size.
    Returns smoothed signal.
    x -- input signal
    window_len -- lenght of smoothing window
    window -- type of window: 'flat', 'hanning', 'hamming',
        'bartlett', 'blackman'
        flat window will produce a moving average smoothing.
    """
    if x.ndim != 1:
        raise ValueError, "smooth only accepts 1 dimension arrays."
    if x.size < window_len:
        raise ValueError, "Input vector needs to be bigger than window
size."
    if window_len < 3:
        return x
    if not window in WINDOWS:
        raise ValueError("Window is one of 'flat', 'hanning', 'hamming', "
            "'bartlett', 'blackman'")
    # adding reflected windows in front and at the end
    s=numpy.r_[x[window_len-1:0:-1], x, x[-1:-window_len:-1]]
    # pick windows type and do averaging
    if window == 'flat': #moving average
```

```
        w = numpy.ones(window_len, 'd')
    else:
        # call appropriate function in numpy
        w = eval('numpy.' + window + '(window_len)')
    # NOTE: length(output) != length(input), to correct this:
    # return y[(window_len/2-1):-(window_len/2)] instead of just y.
    y = numpy.convolve(w/w.sum(), s, mode='valid')
    return y
# Get some evenly spaced numbers over a specified interval.
t = linspace(-4, 4, 100)
# Make some noisy sinusoidal
x = sin(t)
xn = x + randn(len(t))*0.1
# Smooth it
y = smooth(x)
# windows
ws = 31
subplot(211)
plot(ones(ws))
# draw on the same axes
hold(True)
# plot for every windows
for w in WINDOWS[1:]:
    eval('plot('+w+'(ws) )')
# configure axis properties
axis([0, 30, 0, 1.1])
# add legend for every window
```

```
legend(WINDOWS)
title("Smoothing windows")
# add second plot
subplot(212)
# draw original signal
plot(x)
# and signal with added noise
plot(xn)
# smooth signal with noise for every possible windowing algorithm
for w in WINDOWS:
    plot(smooth(xn, 10, w))
# add legend for every graph
l=['original signal', 'signal with noise']
l.extend(WINDOWS)
legend(l)
title("Smoothed signal")
show()
```

从图2-11所示的两个图形中，可以看出窗口算法是如何影响噪声信号的。上面的图形显示了窗口算法，下面的图形显示了每一个相应的结果，包括原始信号、添加了噪声的信号和经过每个算法平滑处理过的信号。可以在代码中试着注释掉一些窗口类型，只保留一到两个窗口，可以更好的理解算法之间的差异。

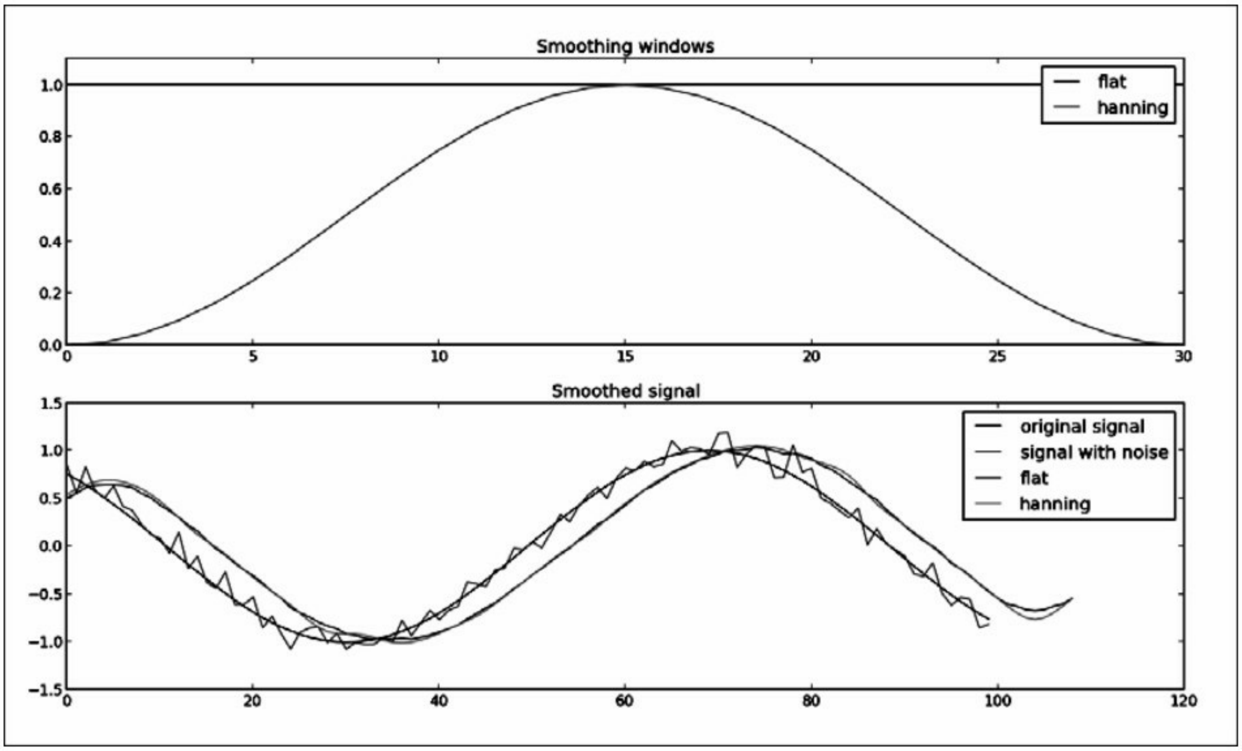


图2-11

2.14.4 补充说明

另一个非常流行的信号平滑处理算法是中值滤波（Median Filter）。中值滤波的中心思想就是逐项地遍历信号，并用相邻信号项的中值替换当前项。这种方法使得滤波处理非常快速，而且对一维数据集合和两维数据集合（例如图像）都适用。

在下面的例子中，我们使用了SciPy信号工具箱中的实现。

```
import numpy as np
import pylab as p
import scipy.signal as signal
# get some linear data
x = np.linspace (0, 1, 101)
# add some noisy signal
```

```
x[3::10] = 1.5
p.plot(x)
p.plot(signal.medfilt(x,3))
p.plot(signal.medfilt(x,5))
p.legend(['original signal', 'length 3','length 5'])
p.show ()
```

从图2-12所示的图形中，可以看到窗口越大，信号和原始信号相比失真越严重，但同时看上去也越平滑。

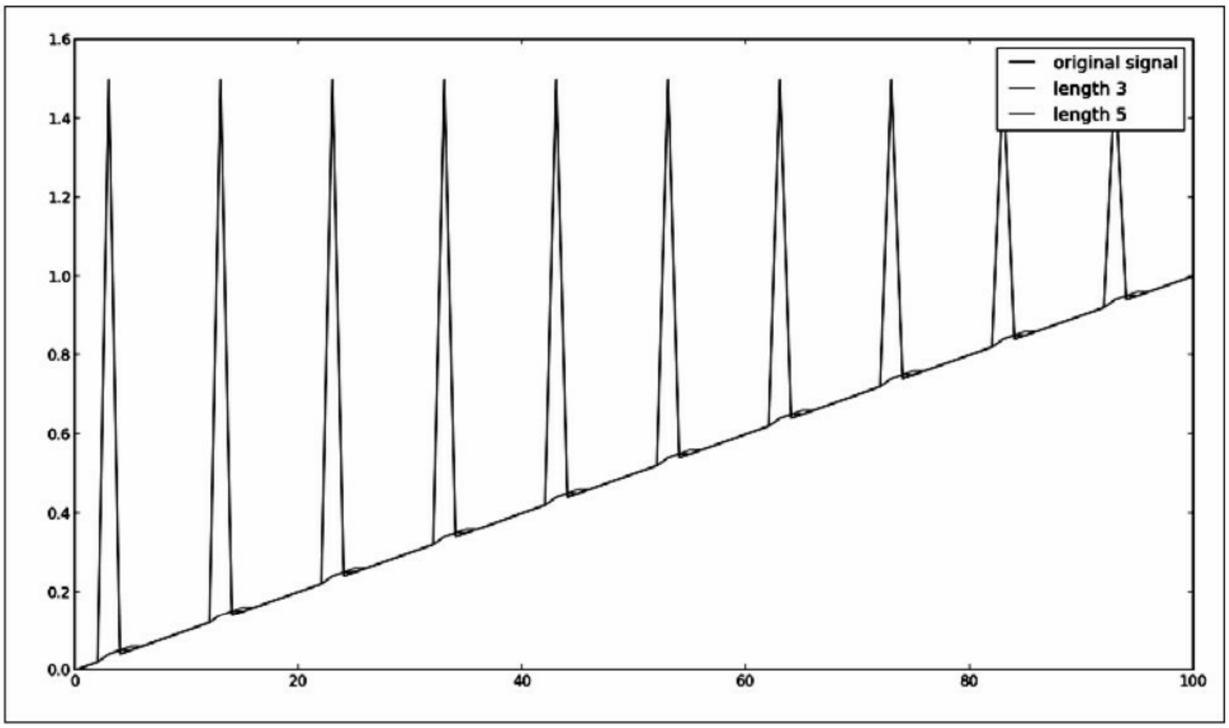


图2-12

许多方法可以对从外部信号源接收到的数据（信号）进行平滑处理，这取决于工作的领域和信号的性质。许多算法都是专门用于某一种特定的信号，可能没有一个通用解决方案普遍适用于所有的情况。

然而，一个非常重要的问题是，“什么时候不应该对信号进行平滑处理？”一个常见的情况是在统计过程（如最小二乘曲线拟合，least-squares curve fitting）之前，因为所有的平滑算法或多或少都会使信号产

生失真，从而改变信号波形。而且，对于真实信号来说，平滑处理的噪声对于真实的信号来说可能是错误的。

注释

[1]. outlier：常译为异常值、离群值或野点，统计学上的一个概念。指样本中的个别值，其数值明显偏离它（或它们）所属样本的其余观测值。本书中统一译为异常值。【译者注】

[2]. 原书为 254，属笔误。

[3]. 原书为：/usr/share/dicts/words

第3章 绘制并定制化图表

本章会详细介绍并展示更多matplotlib的功能，包括以下几方面。

◆ 定义图表类型——柱状图、线形图和堆积柱状图

◆ 绘制简单的正弦图和余弦图

◆ 设置轴的长度和范围

◆ 设置图表的线型、属性和格式化字符串

◆ 设置刻度、刻度标签和网格

◆ 添加图例和注解

◆ 移动轴线到图正中央

◆ 绘制直方图

◆ 绘制误差条形图

◆ 绘制饼图

◆ 绘制带填充区域的图表

◆ 绘制带彩色标记的散点图

3.1 简介

虽然我们已经用matplotlib绘制了一些图表，但并没有详细介绍它们是怎么工作的，怎样设置它们，或者如何用matplotlib做更多的事情。我们研究并练习了大部分基本类型的数据可视化：线形图、柱状图、直方图、饼图以及它们的变形。

Matplotlib是一个强大的工具箱，能满足几乎所有2D和一些3D绘图的需求。通过例子学习matplotlib是其作者推荐的方式。当需要画一个图表时，我们找到一个相似的例子，然后尝试做些改动使其满足我们的要求。因此，我们也打算向你展示一些有用的例子，而且相信能帮助你找到一个和你的需求类似的例子。

3.2 定义图表类型——柱状图、线形图和堆积柱状图

本节将展示基本的图表以及它们的用途。这里介绍的大多数图表都是很常用的，其中有一些是理解数据可视化中更高阶概念的基础。

3.2.1 准备工作

我们从 matplotlib.pyplot 库的一些常用图表入手，采用一些简单的样本数据开始一些基本的绘图操作，为后面几节内容打基础。

3.2.2 操作步骤

我们先在IPython中创建一个简单的图表。IPython非常不错，它能让我们交互式地改变图表并立即看到结果。

1.在命令行键入以下命令来启动IPython。

```
$ ipython --pylab
```

2.然后键入 matplotlib plot 代码。

```
In [1]: plot([1,2,3,2,3,2,2,1])
Out[1]: [<matplotlib.lines.Line2D at 0x412fb50>]
```

图表会显示在一个新打开的窗口中，其默认的外观和一些辅助信息如图3-1所示。

Matplotlib中的基本图表包括以下元素。

◆ x 轴和 y 轴：水平和垂直的轴线。

◆ x 轴和 y 轴刻度：刻度标示坐标轴的分隔，包括最小刻度和最大刻度。

◆ x 轴和 y 轴刻度标签：表示特定坐标轴的值。

◆ 绘图区域：实际绘图的区域。

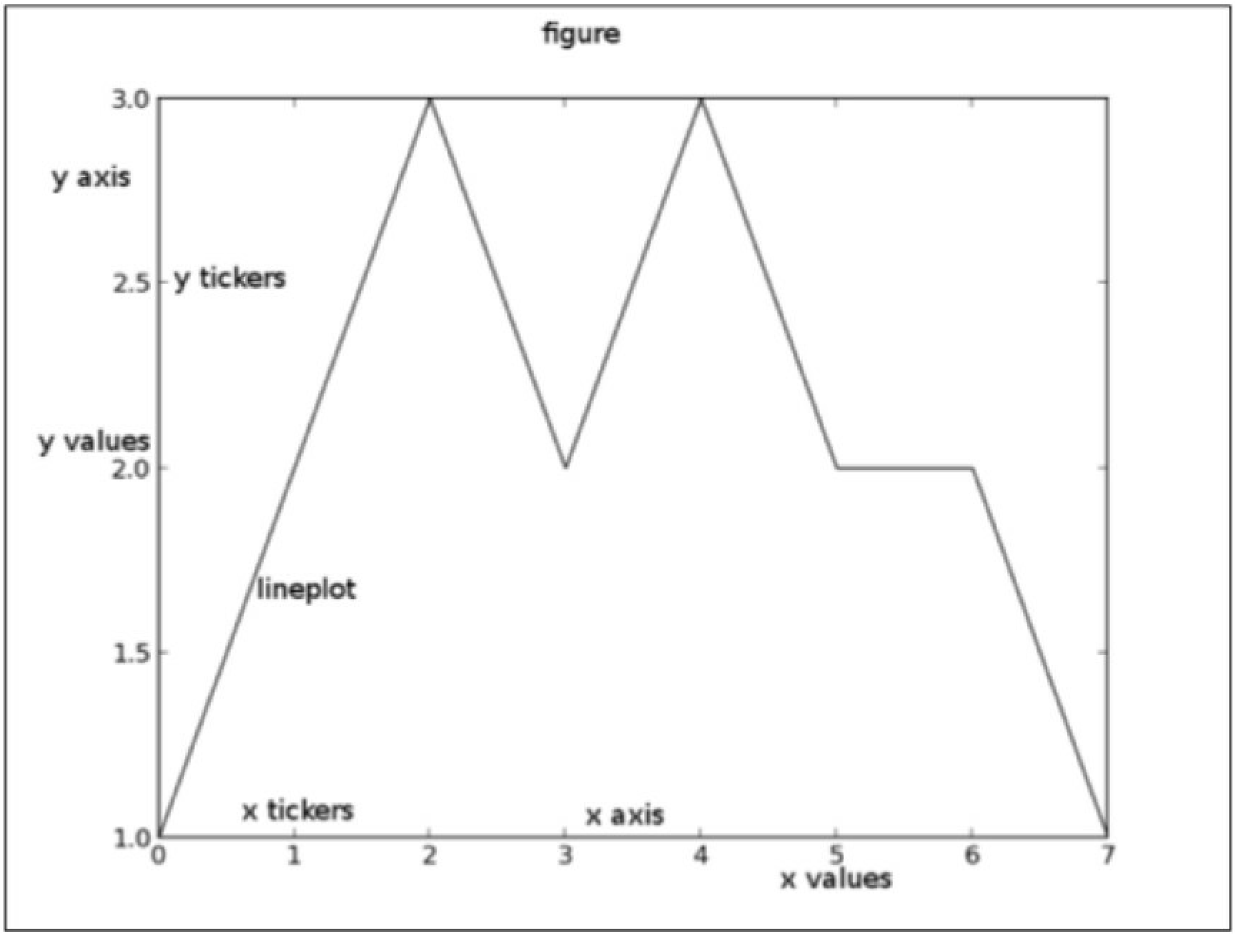


图3-1

你会注意到我们提供给plot()的值是y轴的值。plot()为x轴提供默认值，在这里为从0到7的线性值（y轴对应值1）。

现在，试着通过plot()的第一个参数添加x轴的值，在刚才的IPython会话中键入以下代码。

In [2]: plot([4,3,2,1],[1,2,3,4])

Out[2]: [<matplotlib.lines.Line2D at 0x31444d0>]



注意IPython是如何对输入和输出行进行计数的（In[2]和Out[2]）。这能帮助我们记住我们在当前会话中的位置，并且IPython还提供了更高级的功能，例如把部分会话保存到Python文件中。在数据分析期间，用IPython做原型设计是得到满意方案的最快捷的方式，然后还可以将特定的会话存到文件中，以备将来重新生成相同的图表。

图表会变成如图3-2所示的样子。

由图可知，matplotlib通过扩展y轴来适应新的值范围，并且为了让我们能区分出新的图形，自动改变了第二个线条的颜色。

如果不关闭hold属性（通过调用hold(False)方法），所有接下来的图表都将绘制在相同的坐标轴下。这是IPython的pylab模式的默认行为，然而在编写常规Python脚本中，hold属性默认是关闭的[1]。

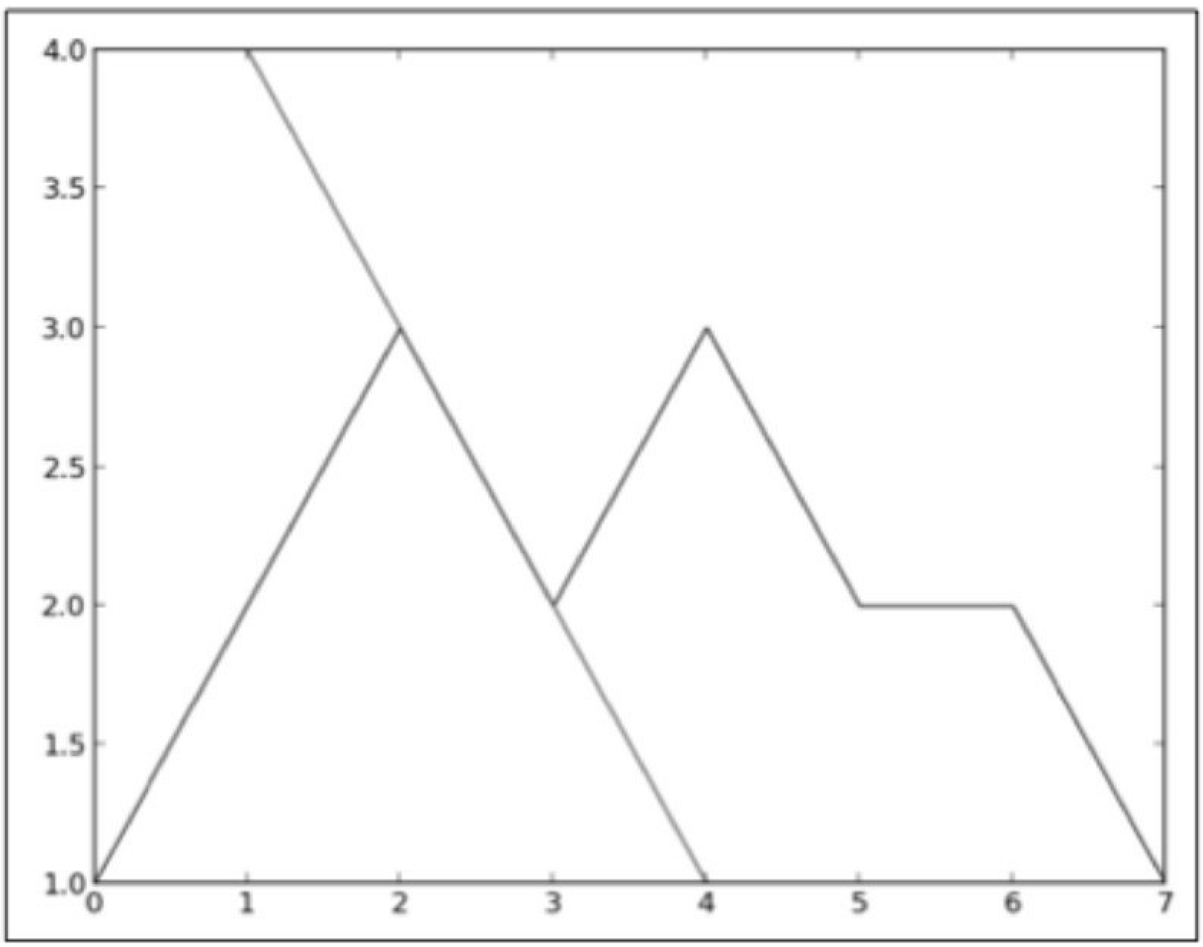


图3-2

让我们基于相同的数据集合多生成一些常见的图表来做一下比较。可以在 IPython 中键入下面的代码，或者在一个单独的Python脚本中运行它。

```
from matplotlib.pyplot import *
# some simple data
x = [1,2,3,4]
y = [5,4,3,2]
# create new figure
figure()
# divide subplots into 2 x 3 grid
# and select #1
subplot(231)
plot(x, y)
# select #2
subplot(232)
bar(x, y)
    # horizontal bar-charts
subplot(233)
barh(x, y)
# create stacked bar charts
subplot(234)
bar(x, y)
# we need more data for stacked bar charts
y1 = [7,8,5,3]
bar(x, y1, bottom=y, color = 'r')
# box plot
```

```
subplot(235)
boxplot(x)
# scatter plot
subplot(236)
scatter(x,y)
show()
```

绘制出来的图表如图3-3所示。

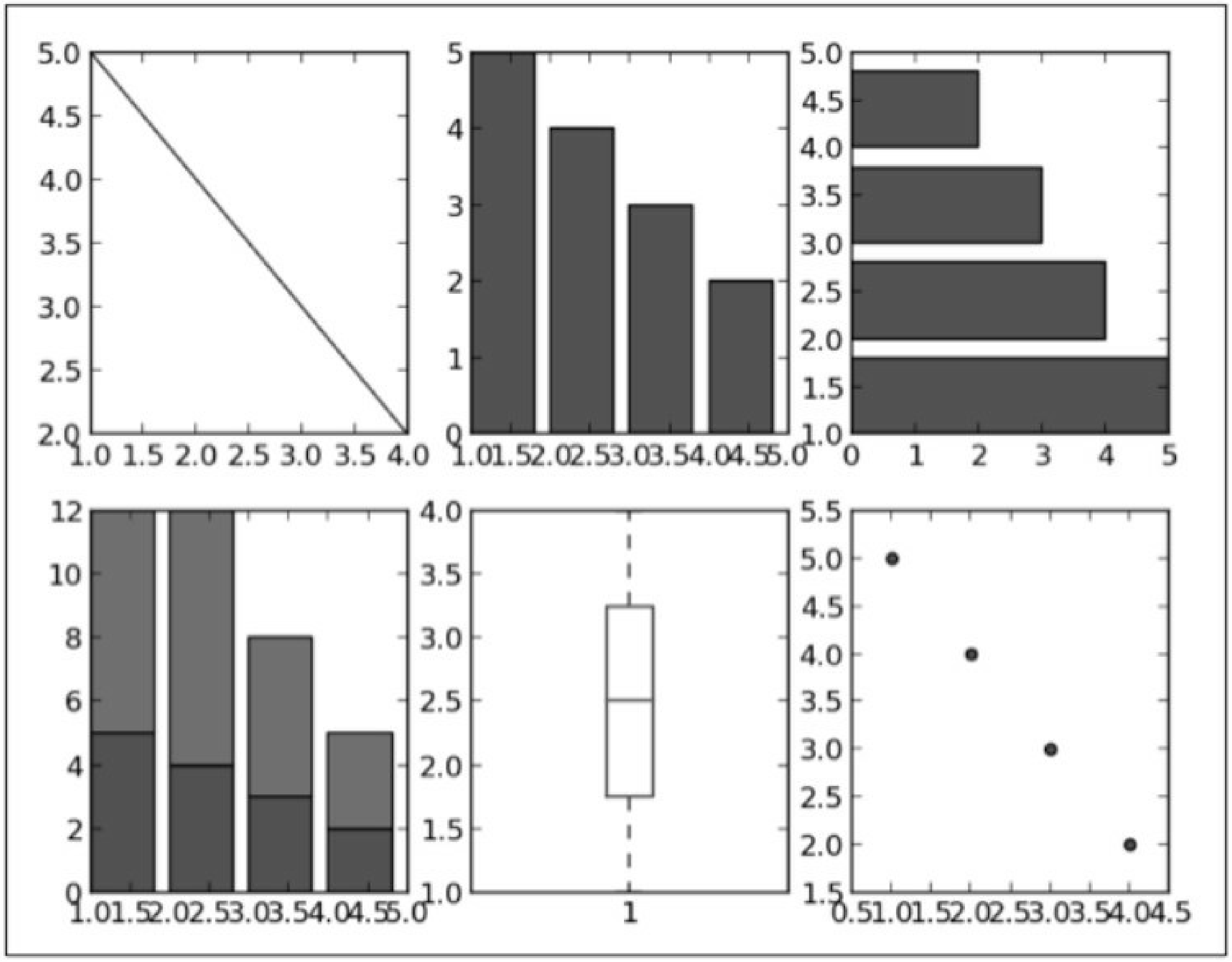


图3-3

3.2.3 工作原理

通过调用figure()方法，我们创建出一个新的图表。如果给方法提供

一个字符串参数，例如 sample charts，这个字符串就会成为窗口的后台标题。如果通过相同的参数（也可以是数字）调用figure()方法，将会激活相应的图表，并且接下来的绘图操作都在此图表中进行。

接下来，调用 subplot(231)方法把图表分割成 2 × 3 的网格。也可以用subplot(3,2,1)这种形式来调用，其中第一个参数是行数，第二个参数是列数，第三个参数表示图形的标号。

接着用几个简单的命令创建垂直柱状图（bar()）和水平柱状图（barh()）。对于堆叠柱状图，我们需要把两个柱状图方法调用连在一起。通过设置参数bottom=y，把第二个柱状图和前一个连接起来形成堆叠柱状图。

通过调用 boxplot()方法可以创建箱线图，图中的箱体从下四分位数延伸到上四分位数，并带有一条中值线。后续我们会继续介绍箱线图。

最后创建了一个散点图来使大家对基于点的数据集合有所了解。当一个数据集合中有成千上万的数据点时，散点图很有可能就更合适了。但这里，我们只是想举例说明相同数据集合的不同展示方式。

3.2.4 补充说明

现在让我们回到箱线图，因为需要解释一下几个最重要的显示选项。

首先，我们可以添加从箱体延伸出来的箱须来展示数据集合的整个范围。箱体和箱须主要用于表现一个或多个数据集合中数据的变化，容易对数据进行对比而且易于理解。在同一个箱线图中可以呈现5种数据。

- 最小值：数据集合的最小值。
- 第二四分位数：其以下为数据集合中较低的 25%数据。
- 中值：数据集合的中值。

◆ 第三四分位数：其以上为数据集合中较高的 25%数据。

◆ 最大值：给定数据集合的最大值。

为了说明一下上述的数据项，在接下来的代码中，我们将用同一个数据集合来绘制箱线图和直方图。

```
from pylab import *
dataset = [113, 115, 119, 121, 124,
   124, 125, 126, 126, 126,
   127, 127, 128, 129, 130,
   130, 131, 132, 133, 136]
subplot(121)
boxplot(dataset, vert=False)
subplot(122)
hist(dataset)
show()
```

生成的图表如图3-4所示。

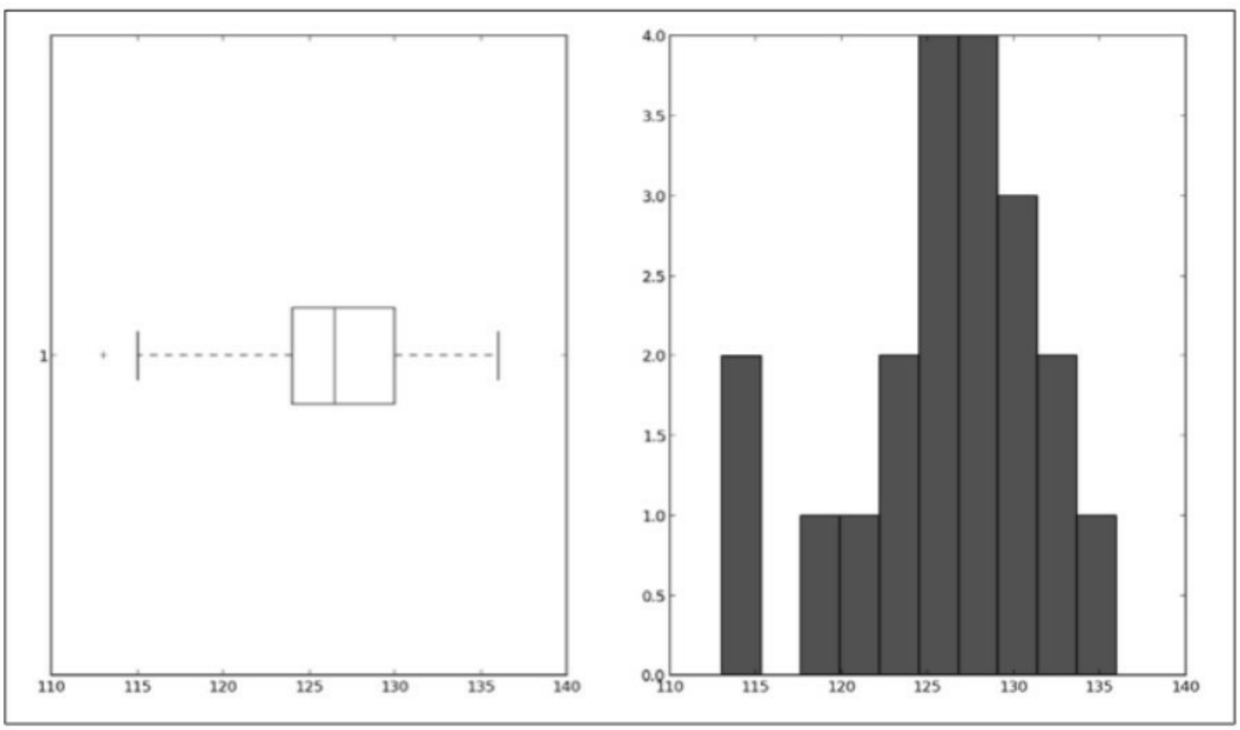


图3-4

通过上述对比，我们可以观察到两种图表在数据展现上的差异。左图呈现了前面提到的五个统计数据，右图（直方图）展示了数据集合在给定范围内的分组情况。

3.3 简单的正弦图和余弦图

本节将复习一下基本的数学函数绘图以及和数学符号相关的知识，比如在标签和绘制的曲线上写上希腊符号。

3.3.1 准备工作

这里我们用到的最多的绘图指令是画线指令，它可以在图表中绘制出给定的(x,y)坐标。

3.3.2 操作步骤

首先，我们对从-Pi[2]到Pi之间具有相同的线性距离的256个点来计算正弦值和余弦值，然后把sin(x)值和cos(x)值在同一个图表中绘制出来。

```
import matplotlib.pyplot as pl
import numpy as np
x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
y = np.cos(x)
y1 = np.sin(x)
pl.plot(x,y)
pl.plot(x,y1)
pl.show()
```

生成的图表如图3-5所示。

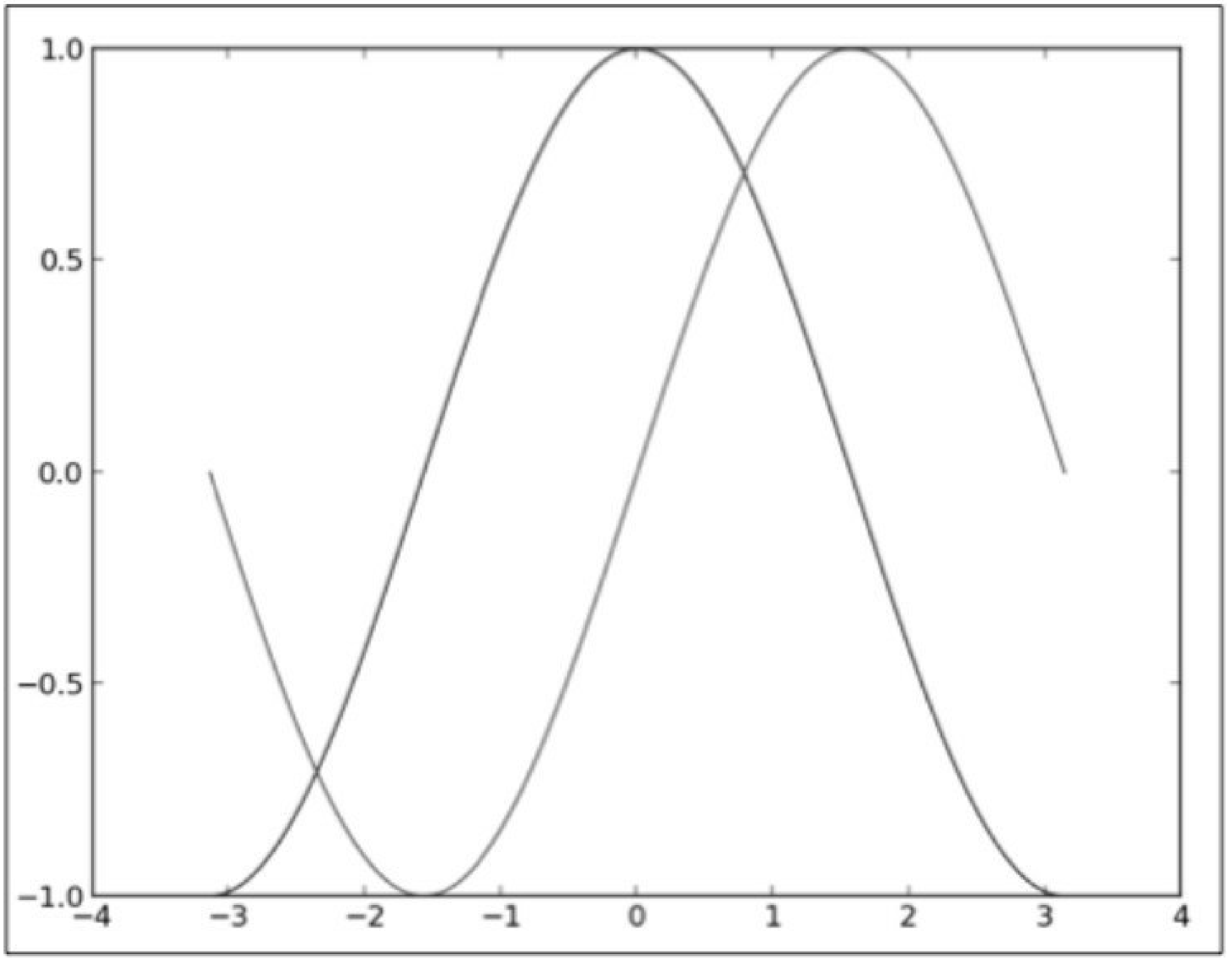


图3-5

以这个简单图表为基础，可以进一步定制化来添加更多的信息，并且让坐标轴及其边界更精确些。

```
from pylab import *
import numpy as np
# generate uniformly distributed
# 256 points from -pi to pi, inclusive
x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
# these are vectorised versions
# of math.cos, and math.sin in built-in Python maths
# compute cos for every x
```

```
y = np.cos(x)
# compute sin for every x
y1 = np.sin(x)
# plot cos
plot(x, y)
# plot sin
plot(x, y1)
# define plot title
title("Functions $\sin$ and $\cos$")
# set x limit
xlim(-3.0, 3.0)
# set y limit
ylim(-1.0, 1.0)
# format ticks at specific values
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
    [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
yticks([-1, 0, +1],
    [r'$-1$', r'$0$', r'$+1$'])
show()
```

生成的图表会比较漂亮，如图3-6所示。

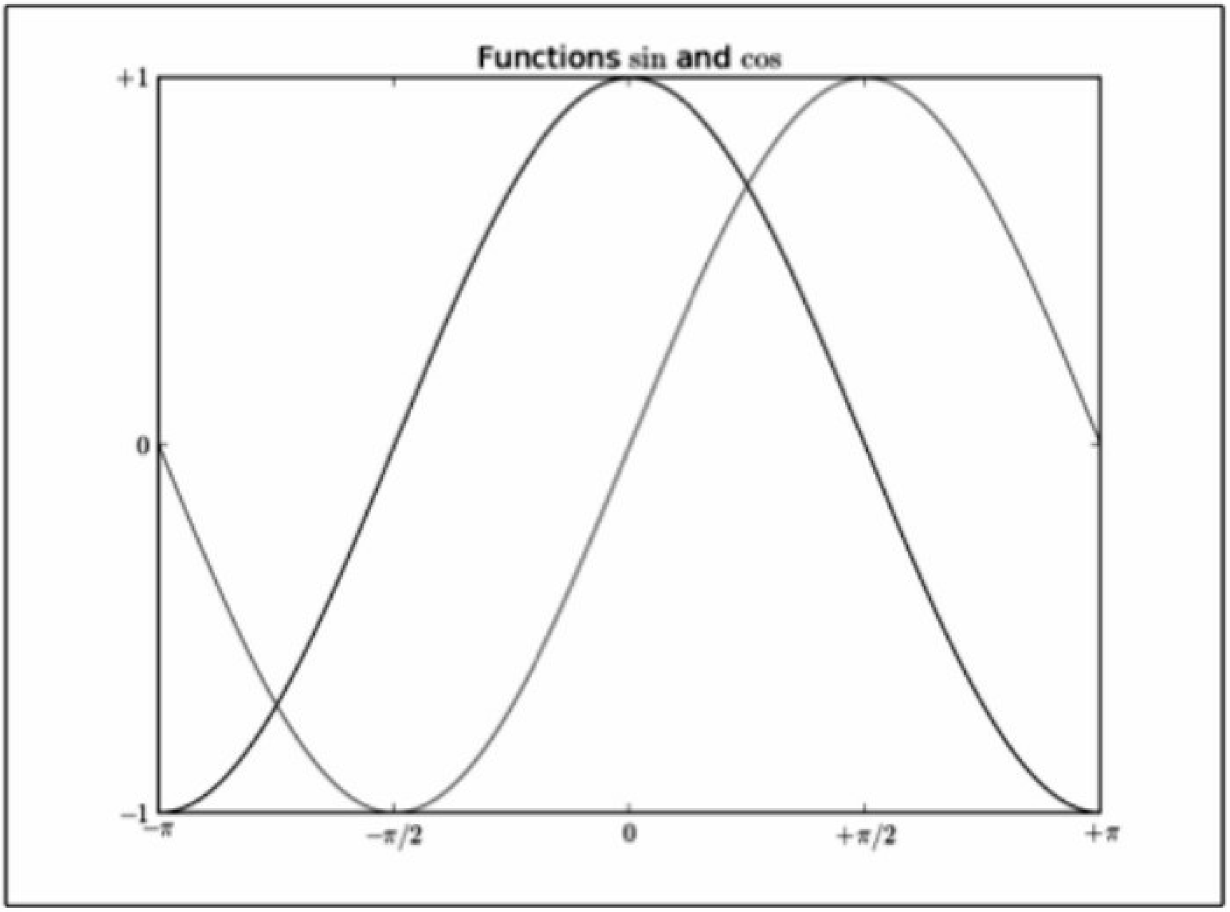


图3-6

上述代码中，用如$\sin$，或$-\pi$的表达式在图表中写上希腊字母。在接下来的章节中会进一步介绍这种LaTex语法。这里，我们仅仅为了说明让你的数学图表更可读是多么的简单。

3.4 设置坐标轴长度和范围

本节将演示一些与坐标轴的范围和长度相关的非常有用的属性，可以在matplotlib中配置这些属性。

3.4.1 准备工作

本节中的内容将用IPython来演示：

$ ipython --pylab

3.4.2 操作步骤

首先，让我们用坐标轴的不同属性来做个实验。调用不带参数的axis()方法将返回坐标轴的默认值。

In [1]: axis()

Out[1]: (0.0, 1.0, 0.0, 1.0)

注意，如果是在交互模式下，并且使用了窗口后端，将会显示一个只有坐标轴的空白图。

这里的值分别表示xmin、xmax、ymin和ymax。同样，我们可以设置x轴和y轴的值。

In [2]: l = [-1, 1, -10, 10]

In [3]: axis(l)

Out[3]: [-1, 1, -10, 10]

再次说明一下，在交互模式下会更新相同的图形。而且，也可以通过关键字参数（**kwargs）单独更新某一个参数值，例如仅把xmax设置为某个值。

3.4.3 工作原理

如果不使用axis()或者其他参数设置，matplotlib会自动使用最小值，刚好可以让我们在一个图中看到所有的数据点。如果设置 axis()的范围比数据集合中的最大值小，matplotlib按照设置执行，这样就无法在图中看到所有的数据点。这可能会引起困惑甚至是错误，因为我们认为我们看到了绘制的所有东西。避免这种情况发生的一种方法是调用autoscale()(matplotlib.pyplot.autoscale())[3]方法，该方法会计算坐标轴的最佳大小以适应数据的显示。

如果想向相同图形中添加新的坐标轴，可以调用matplotlib.pyplot.axes()方法。我们通常会在方法中传入一些属性，例如rect，归一化单位（0，1）下的left、bottom，width、height四个属性，或者axisbg，该参数指定坐标轴的背景颜色。

还有其他一些参数允许我们对新添加的坐标轴进行设置，如sharex/sharey参数，接收其他坐标轴的值并让当前坐标轴（x/y）共享相同的值；或者 polar 参数，指定是否使用极坐标轴（polar axes）。

添加新坐标轴是有用的，例如，如果需要几个不同的视图来表达相同的数据的不同属性值，这就需要在一张图中组合显示多个图表。

如果只想对当前图形添加一条线，可以调用matplotlib.pyploy.axhline()或者matplotlib.pyplot.axvline()。axhline()和axvline()方法会根据给定的x和y值相应地绘制出相对于坐标轴的水平线和垂直线。这两个方法的参数很相似，axhline()方法比较重要的参数是y 向位置、xmin 和 xmax，axvline()方法比较重要的参数是 x向位置、ymin和ymax。

让我们在图表中看一下，继续在相同的IPython会话中操作。

```
In [3]: axhline()
Out[3]: <matplotlib.lines.Line2D at 0x414ecd0>
```

```
In [4]: axvline()
Out[4]: <matplotlib.lines.Line2D at 0x4152490>
In [5]: axhline(4)
Out[5]: <matplotlib.lines.Line2D at 0x4152850>
```

得到如图3-7所示的图形。

在这里，我们看到调用这些方法时如果不传入参数，就会使用默认值。axhline()方法绘制了一条y=0的水平线，axvline()绘制了一条x=0的垂直线。

类似的，另外两个相关的方法允许我们添加一个跨坐标轴的水平带（矩形），它们是matplotlib.pyplot.axhspan()和matplotlib.pyplot.axvspan()[4]。axhspan()方法必需的ymin和ymax参数指定了水平带的宽度。同理，axvspan()方法必需的xmin和xmax参数指定了垂直带的宽度。

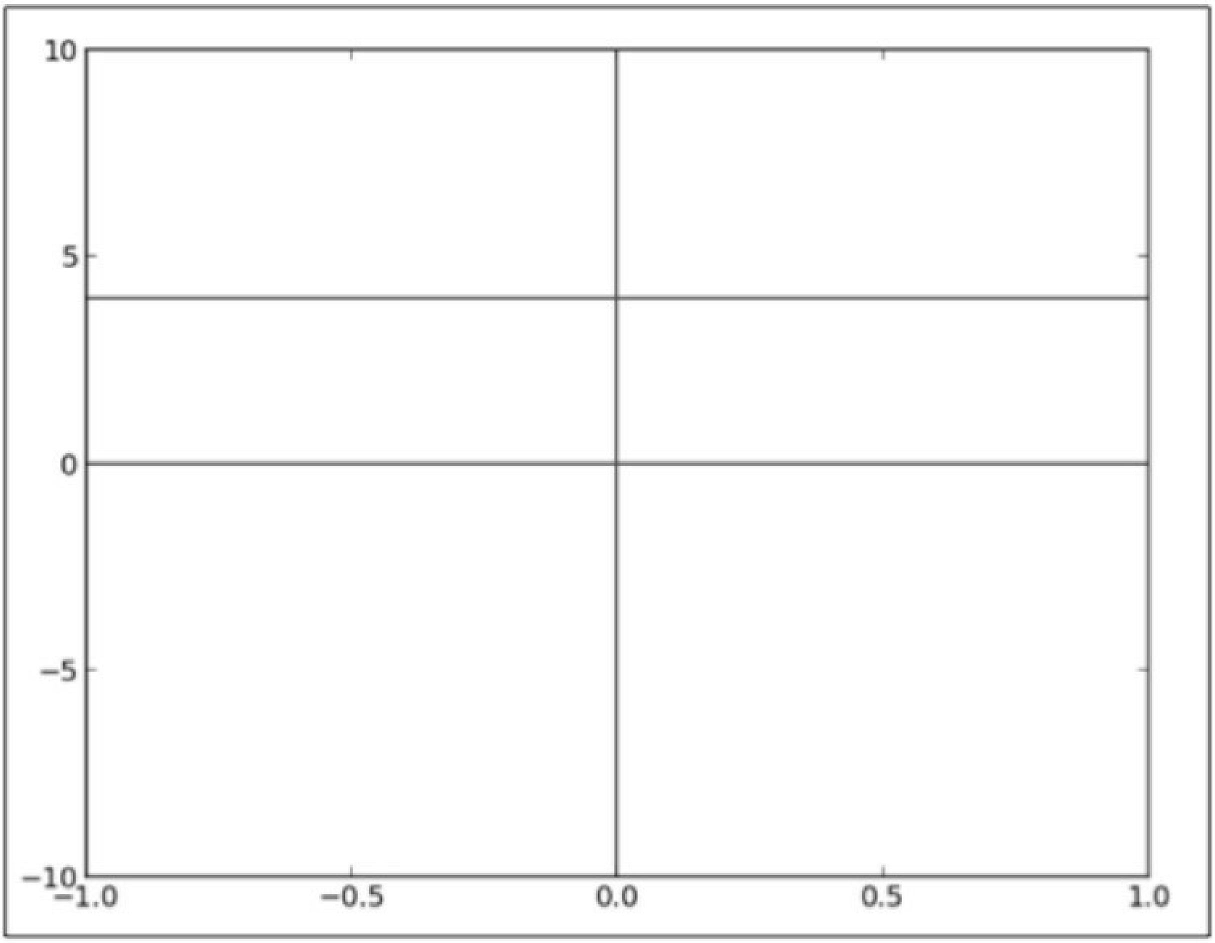


图3-7

3.4.4 补充说明

图形中的网格属性默认是关闭的，但可以很简单地打开和定制化。不带参数调用matplotlib.pyplot.grid()会切换网格的显示状态。另外一些控制参数如下。

◆ which：指定绘制的网格刻度类型（major、minor 或者 both）。

◆ axis：指定绘制哪组网格线（both、x 或者 y）。

坐标轴通常由 matplotlib.pyplot.axis()控制。坐标轴在内部实现上由几个Python类来表示。其中一个父类是matplotlib.axes.Axes，包含了操作坐标轴的大多数方法。单独一个坐标轴由matplotlib.axis.Axis类来表

示，matplotlib.axis. XAxis表示x轴，matplotlib.axis.YAxis表示y轴。

在做本节练习的过程中，我们不需要这些类。但是重要的是，如果我们对更高级的坐标轴控制感兴趣，并且在 matplot.pyplot 命名空间下已经不能满足需求的时候，我们知道去哪里找。

3.5 设置图表的线型、属性和格式化字符串

本节将演示如何改变线的各种属性，如线条风格、颜色或者宽度。根据要表达的信息合理地设置线型并明显地区分目标受众（受众如果是年轻群体，可以使用比较生动的颜色；如果是上年纪的人，可能需要使用对比更强烈的颜色）能让图表给观众留下非常深刻的印象。

3.5.1 准备工作

虽然我们强调美化图表的重要性，但首先我们要学会怎样做。

如果你对颜色匹配不是很敏感，这里有一些免费和商业的在线工具可以为你生成颜色集。Colorbrewer2是最有名的工具之一，其链接为http://colorbrewer2.org/。

已经有一些针对数据可视化中颜色使用方面的严谨的研究在进行中，但是解释其理论已经超出了本书的范围。如果你每天要与更高级的可视化打交道，应当阅读一下与这些话题相关的资料。

3.5.2 操作步骤

首先学习如何改变线的属性，可以通过不同的方法来改变图表中的线条。

第一个最常用的方式是向方法传入关键字参数来指定线型，例如plot()方法。

plot(x, y, linewidth=1.5)

对 plot()方法的调用返回一个线条的实例（matplotlib.lines.Line2D），可以在这个实例上用一系列的setter方法来

设置不同的属性。

```
line, = plot(x, y)
line.set_linewidth(1.5)
```

使用过MATLAB©的人会更习惯使用第三种方式配置线条属性——使用setp()方法。

```
lines = plot(x, y)
setp(lines, 'linewidth', 1.5)
```

另一种使用setp的方式是。

```
setp(lines, linewidth=1.5)
```

不管你喜欢用哪种方式来配置线型，选择一种并在整个项目中（或至少在一个文件中）保持一致。这样，当你（或者别人）将来再看代码时，会更容易明白和修改它。

3.5.3 工作原理

用来改变线条的所有属性都包含在matplotlib.lines.Line2D类中，表3-1中列举了一些属性。

表3-1

<table>
<tr><th>属　　性</th><th>类　　型</th><th>描　　述</th></tr>
<tr><td>alpha</td><td>浮点值</td><td>alpha 值用来设置混色，并不是所有后端都支持</td></tr>
<tr><td>color 或 c</td><td>任意 matplotlib 颜色</td><td>设置线条颜色</td></tr>
<tr><td>dashes</td><td>以点为单位的 on/off 序列①</td><td>设置破折号序列，如果 seq 为空或者如果 seq=[None, None]，linestyle 将被设置为 solid</td></tr>
<tr><td>label</td><td>任意字符串</td><td>为图例设置标签值</td></tr>
<tr><td>linestyle 或 ls</td><td>['-' | '--' | '-.' | ':' | 'steps' | ...]</td><td>设置线条风格（也接受 drawstyles 的值）</td></tr>
<tr><td>linewidth 或 lw</td><td>以点为单位的浮点值</td><td>设置以点为单位的线宽</td></tr>
<tr><td>marker</td><td>[7 | 4 | 5 | 6 | 'o' | 'D' | 'h' | 'H' | '_' | '' | 'None' | ' ' | None | '8' | 'p' | ',' | '+' | '.' | 's' | '*' | 'd' | 3 | 0 | 1 | 2 | '1' | '3' | '4' | '2' | 'v' | '<' | '>' | '^' | '|' | 'x' | '$...$' | tuple | Nx2 array]</td><td>设置线条标记</td></tr>
<tr><td>markeredgecolor 或 mec</td><td>任意 matplotlib 颜色</td><td>设置标记的边缘颜色</td></tr>
<tr><td>markeredgewidth 或 mew</td><td>以点为单位的浮点值</td><td>设置以点为单位的标记边缘宽度</td></tr>
<tr><td>markerfacecolor 或 mfc</td><td>任意 matplotlib 颜色</td><td>设置标记的颜色</td></tr>
<tr><td>markersize 或 ms</td><td>浮点值</td><td>设置以点为单位的标记大小</td></tr>
<tr><td>solid_capstyle</td><td>['butt' | 'round' | 'projecting']</td><td>设置实线的线端风格</td></tr>
<tr><td>solid_joinstyle</td><td>['miter' | 'round' | 'bevel']</td><td>设置实线的连接风格</td></tr>
<tr><td>visible</td><td>[True | False]</td><td>显示或隐藏 artist</td></tr>
<tr><td>xdata</td><td>np.array</td><td>设置 x 的 np.array 值</td></tr>
</table>

设置破折号序列各段的宽度。举个例子：如果 dashes 序列为[1,5,10]，那么第一段线为 1 个点的宽度，接下来的空白区为5个点的宽度，再接下来的线为10个点的宽度；以此类推，当序列到最后一个值

后，再按第一个值设定下一段的宽度。

续表

属　　性	类　　型	描　　述
ydata	np.array	设置 y 的 np.array 值
Zorder	任意数字	为 artist 设置 z 轴顺序，低 Zorder 的 artist 会先绘制 如果在屏幕上 x 轴水平向右，y 轴垂直向上，那么 z 轴将指向观察者。这样，0 表示在屏幕上，1 表示上面的一层，以此类推。

下表展示了一些线条风格。

线条风格	描　　述	线条风格	描　　述
'-'	实线	':'	虚线
'--'	破折线	'None', ' ', ''	什么都不画
'-.'	点划线		

下图（表格）展示了线条的标记：

标　　记	描　　述	标　　记	描　　述
'o'	圆圈	'.'	点
'D'	菱形	's'	正方形
'h'	六边形 1	'*'	星号
'H'	六边形 2	'd'	小菱形
'_'	水平线	'v'	一角朝下的三角形
'', 'None', ' ', None	无	'<'	一角朝左的三角形
'8'	八边形	'>'	一角朝右的三角形
'p'	五边形	'^'	一角朝上的三角形
','	像素	'|'	竖线
'+'	加号	'x'	X

颜色

可以通过调用matplotlib.pyplot.colors()得到matplotlib支持的所有颜色，如表3-2所示。

表3-2

别　　名	颜　　色	别　　名	颜　　色
b	蓝色	g	绿色

续表

别　　名	颜　　色	别　　名	颜　　色
r	红色	y	黄色
c	青色	k	黑色
m	洋红色	w	白色

这些颜色可以被用在matplotlib中带颜色参数的不同的方法中。

如果这些基本的颜色不够用——随着进一步深入，肯定会不够用——可以用两种其他方式来定义颜色值。一种方法是使用HTML十六进制字符串。

color = '#eeefff'

还可以使用合法的 HTML 颜色名字（'red', 'chartreuse'）。也可以传入一个归一化到[0, 1]的 RGB 元组。

color = (0.3, 0.3, 0.4)

很多方法接受颜色参数，如title()。

title('Title in a custom color', color='#123456')

背景色

通过向如matplotlib.pyplot.axes()或者matplotlib.pyplot.subplot()这样的方法提供一个axisbg参数，我们可以指定坐标轴的背景色。

subplot(111, axisbg=(0.1843, 0.3098, 0.3098))

3.6 设置刻度、刻度标签和网格

本节继续学习如何设置坐标轴和线条属性，并向图形和图表中添加更多的数据。

3.6.1 准备工作

让我们先了解一下图形（figure）和子区[5]（subplots）。

在 matplotlib 中，调用 figure()会显式地创建一个图形，表示一个图形用户界面窗口。通过调用　plot()或类似的方法会隐式地创建图形。这对于简单的图表没有问题，但是对于更高级的应用，能显示创建图形并得到实例的引用是非常有用的。

一个图形包括一个或多个子区。子区能以规则网格的方式排列plot。我们已经使用过subplot()方法，在调用时指定所有plot的行数和列数以及要操作的plot的序号。

如果需要更多的控制，我们需要使用matplotlib.axes.Axes类的坐标轴实例。这样可以把plot放置在图形窗口中的任意位置，例如可以把一个小的plot放在一个大的plot中。

3.6.2 操作步骤

刻度是图形的一部分，由刻度定位器（tick　locator）——指定刻度所在的位置——和刻度格式器（tick　formatter）——指定刻度显示的样式——组成。刻度有主刻度（major ticks）和次刻度（minor ticks），默认不显示次刻度。更重要的是，主刻度和次刻度可以被独立地指定位置和格式化。

我们可以使用 matplotlib.pyplot.locator_params()方法控制刻度定位器的行为。尽管刻度位置通常会自动被确定下来，我们还是可以控制刻度的数目，以及在 plot比较小时使用一个紧凑视图（tight view）。

```
from pylab import *
# get current axis
ax = gca()
# set view to tight, and maximum number of tick intervals to 10
ax.locator_params(tight=True, nbins = 10)
# generate 100 normal distribution values
ax.plot(np.random.normal(10, .1, 100))
show()
```

生成如图3-8所示的图表。

我们可以看到 x 轴和 y 轴是如何被切分的，以及数值是如何显示的。我们也可以用locator类完成相同的设置。下面代码的意思是设置主定位器为10的倍数。

```
ax.xaxis.set_major_locator(matplotlib.ticker.MultipleLocator(10))
```

刻度格式器的配置非常简单。格式器规定了值（通常是数字）的显示方式。例如，用matplotlib.ticker.FormatStrFormatter可以方便地指定'%2.1f'或者'%1.1f cm'的字符串格式作为刻度标签。

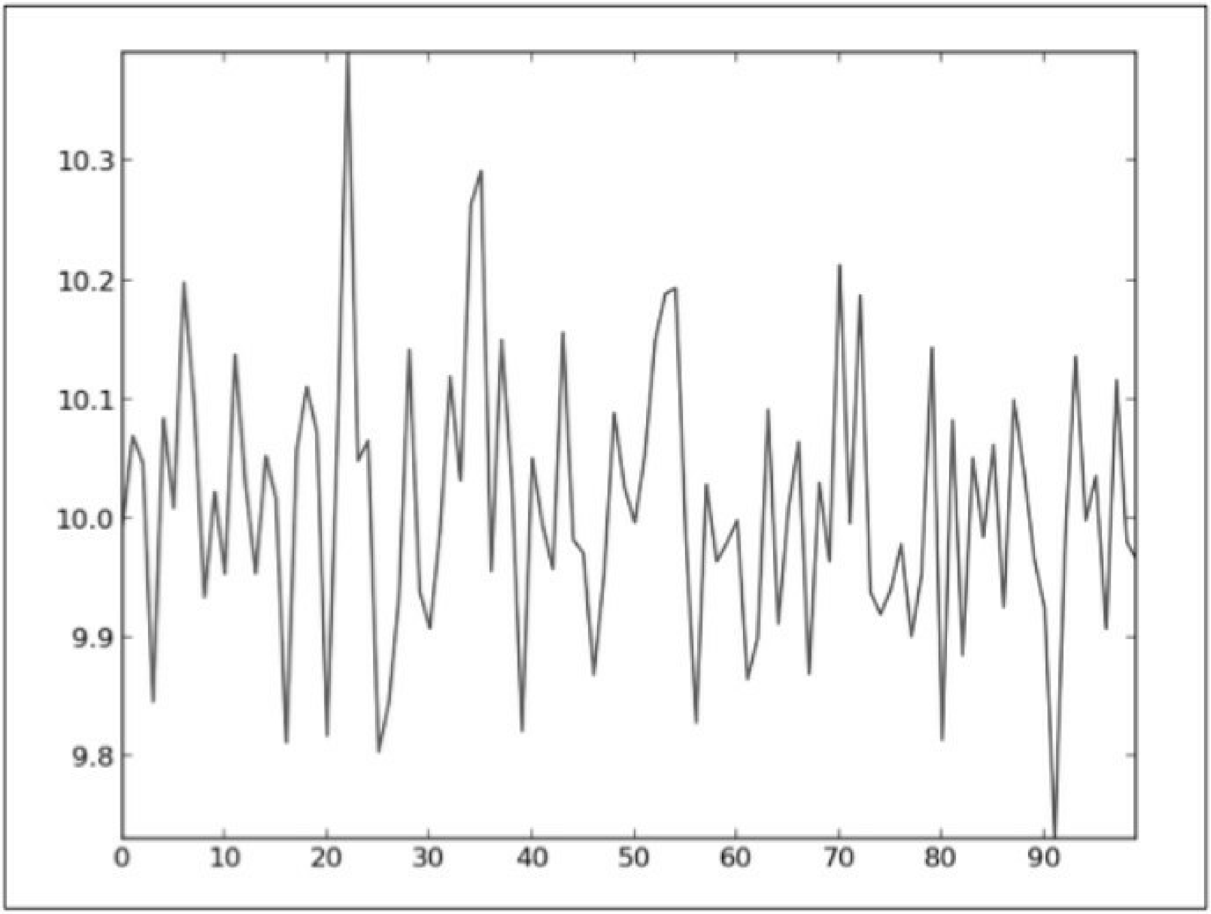
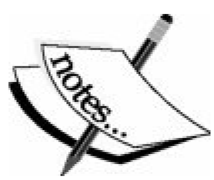


图3-8

让我们看一个使用dates模块的例子。



matplotlib 用浮点值表示日期，其值为从 0001-01-01 UTC 起的天数加 1。因此，0001-01-01 UTC 06:00 的值为 1.25 。

然后，我们可以用 matplotlib.dates.date2num()、matplotlib.dates.num2 date()和matplotlib.dates.drange()这样的helper方法对日期进行不同形式的转换。

再看一个例子：

from pylab import *

```
import matplotlib as mpl
import datetime
fig = figure()
# get current axis
ax = gca()
# set some daterange
start = datetime.datetime(2013, 01, 01)
stop = datetime.datetime(2013, 12, 31)
delta = datetime.timedelta(days = 1)
# convert dates for matplotlib
dates = mpl.dates.drange(start, stop, delta)
# generate some random values
values = np.random.rand(len(dates))
ax = gca()
# create plot with dates
ax.plot_date(dates, values, linestyle='-', marker='')
# specify formater
date_format = mpl.dates.DateFormatter('%Y-%m-%d')
# apply formater
ax.xaxis.set_major_formatter(date_format)
# autoformat date labels
# rotates labels by 30 degrees by default
# use rotate param to specify different rotation degree
# use bottom param to give more room to date labels
fig.autofmt_xdate()
show()
```

上面的代码生成如图3-9所示的图形。

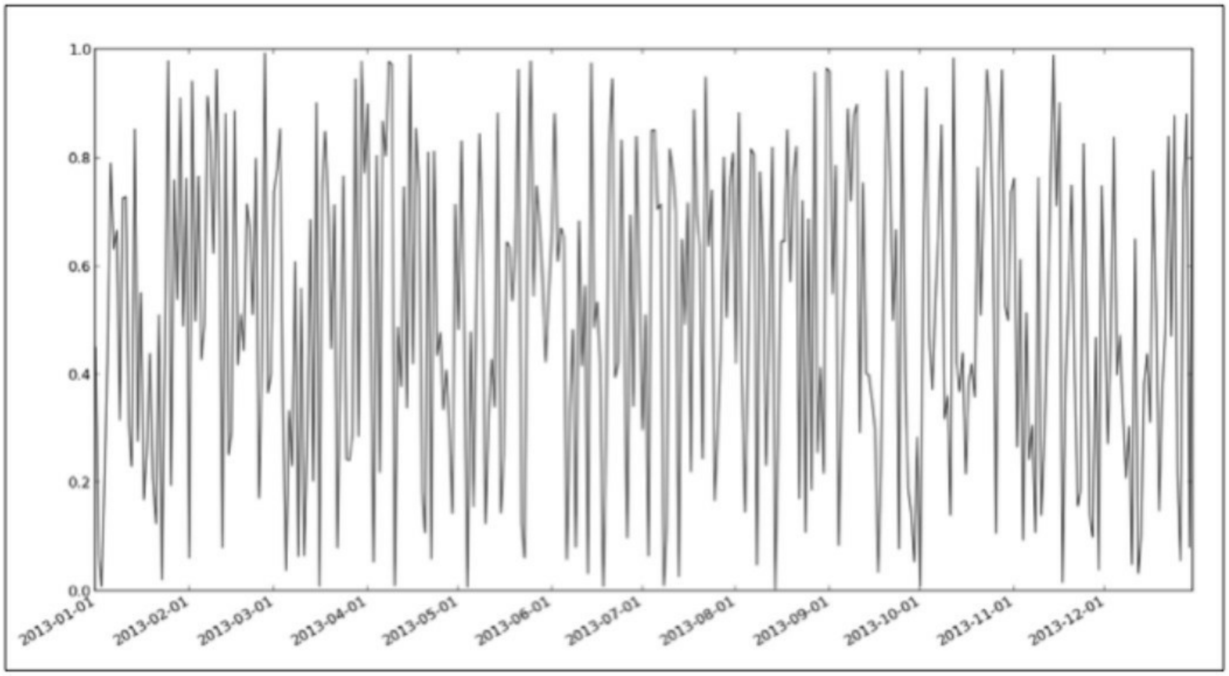

1.0
0.8
0.6
0.4
0.2
0.0
2013-01-01
2013-02-01
2013-03-01
2013-04-01
2013-05-01
2013-06-01
2013-07-01
2013-08-01
2013-09-01
2013-10-01
2013-11-01
2013-12-01

图3-9

3.7 添加图例和注解

图例和注解清晰连贯地解释了数据图表的内容。通过给每个plot添加一个关于所显示数据的简短描述，能让读者（观察者）更容易理解。本节将演示如何对图形中的特定点进行注解，以及如何创建和放置数据图例。

3.7.1 准备工作

请问，有多少次你看着一个图表却不知道它要表达什么？大多数情况下，报纸和其他一些日刊或者周刊中的图表都没有恰当的图例，这让读者对图表有了各种解读，因此会让读者产生歧义，从而增加了出错的可能性。

3.7.2 操作步骤

让我们用下面的例子来演示一下如何添加图例和注解。

```
from matplotlib.pyplot import *
# generate different normal distributions
x1 = np.random.normal(30, 3, 100)
x2 = np.random.normal(20, 2, 100)
x3 = np.random.normal(10, 3, 100)
# plot them
plot(x1, label='plot')
plot(x2, label='2nd plot')
plot(x3, label='last plot')
```

```
# generate a legend box
legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
   ncol=3, mode="expand", borderaxespad=0.)
# annotate an important value
annotate("Important value", (55,20), xycoords='data',
   xytext=(5, 38),
   arrowprops=dict(arrowstyle='->'))
show()
```

上述代码生成如图3-10所示的图表。

我们所做的是为每个plot指定了一个字符串标签，这样legend()会把它们添加到图例框中。

我们通过指定loc参数确定图例框的位置。这个参数是可选的，但是为了不让图例框覆盖图表中的线，我们想为其指定一个位置。

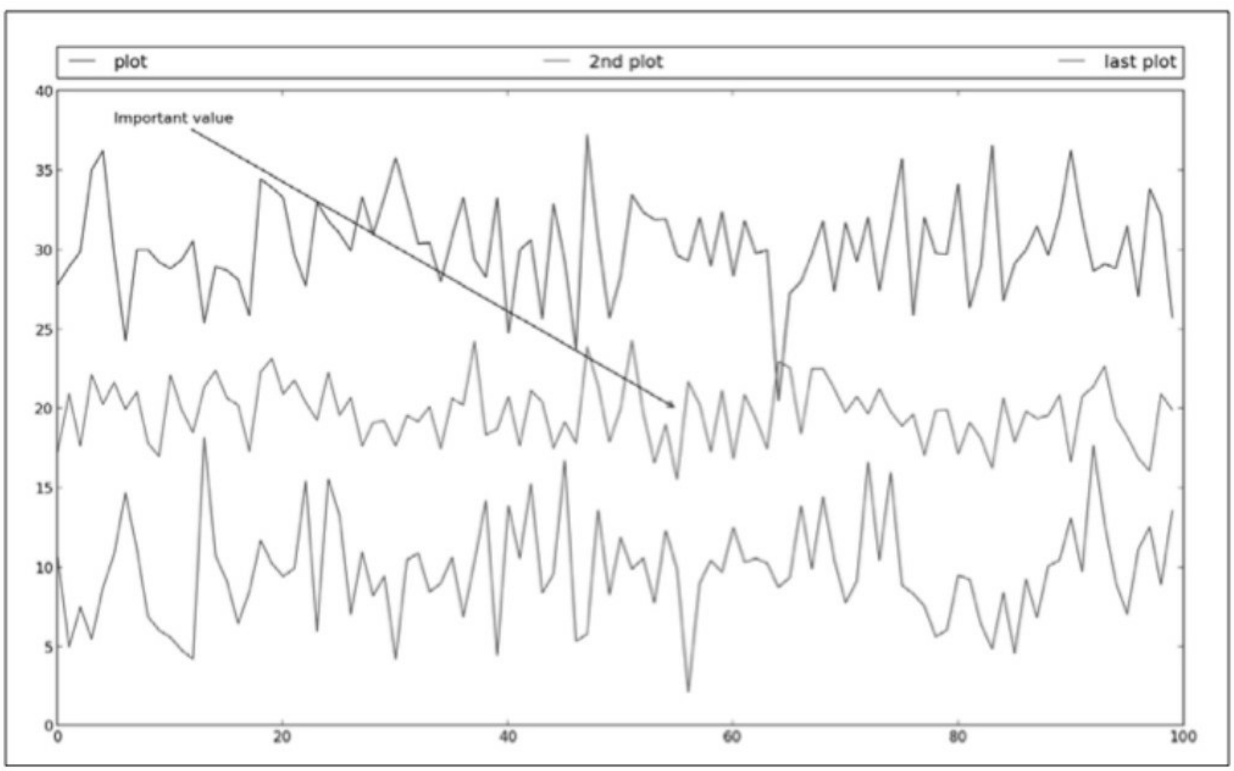


图3-10

3.7.3 工作原理

表3-3列出了所有位置参数。

表3-3

字　符　串	数　　值	字　符　串	数　　值
upper right	1	center left	6
upper left	2	center right	7
lower left	3	lower center	8
lower right	4	upper center	9
right	5	center	10

如果不想在图例中显示标签，可以将标签设置为_nolegend_。

对于上例中的图例，我们设置列数为 ncol=3，设置位置为 lower left。指定边界框（bbox_to_anchor）的起始位置为(0.0, 1.02)，并且设置宽度为1，高度为0.102。这些值都是基于归一化轴坐标系。参数mode可以设置为None或者expand，当为expand时，图例框会水平扩展至整个坐标轴区域。参数borderaxespad指定了坐标轴和图例边界之间的间距。

对于注解，我们在plot中为xy[6]坐标位置的数据点添加了一个字符串描述。通过设置xycoord = 'data'，可以指定注解和数据使用相同的坐标系。注解文本的起始位置通过xytext指定。

箭头由xytext指向xy坐标位置。arrowprops字典中定义了很多箭头属性。在这个例子中，我们用arrowstyle来指定箭头的风格。

3.8 移动轴线到图中央

本节将演示如何移动轴线到图中央。

轴线定义了数据区域的边界，把坐标轴刻度标记连接起来。一共有四个轴线，可以把它们放置在任何位置。默认情况下，它们被放置在坐标轴的边界，因此我们会看到数据图表有一个框。

3.8.1 操作步骤

为了把轴线移到图中央，需要把其中两个轴线隐藏起来（设置color为none）。然后，移动另外两个到坐标（0，0）。坐标为数据空间坐标。

做法如下面代码所示。

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-np.pi, np.pi, 500, endpoint=True)
y = np.sin(x)
plt.plot(x, y)
ax = plt.gca()
# hide two spines
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
# move bottom and left spine to 0,0
ax.spines['bottom'].set_position(('data',0))
ax.spines['left'].set_position(('data',0))
```

```
# move ticks positions
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.show()
```

生成如图3-11所示的图形。

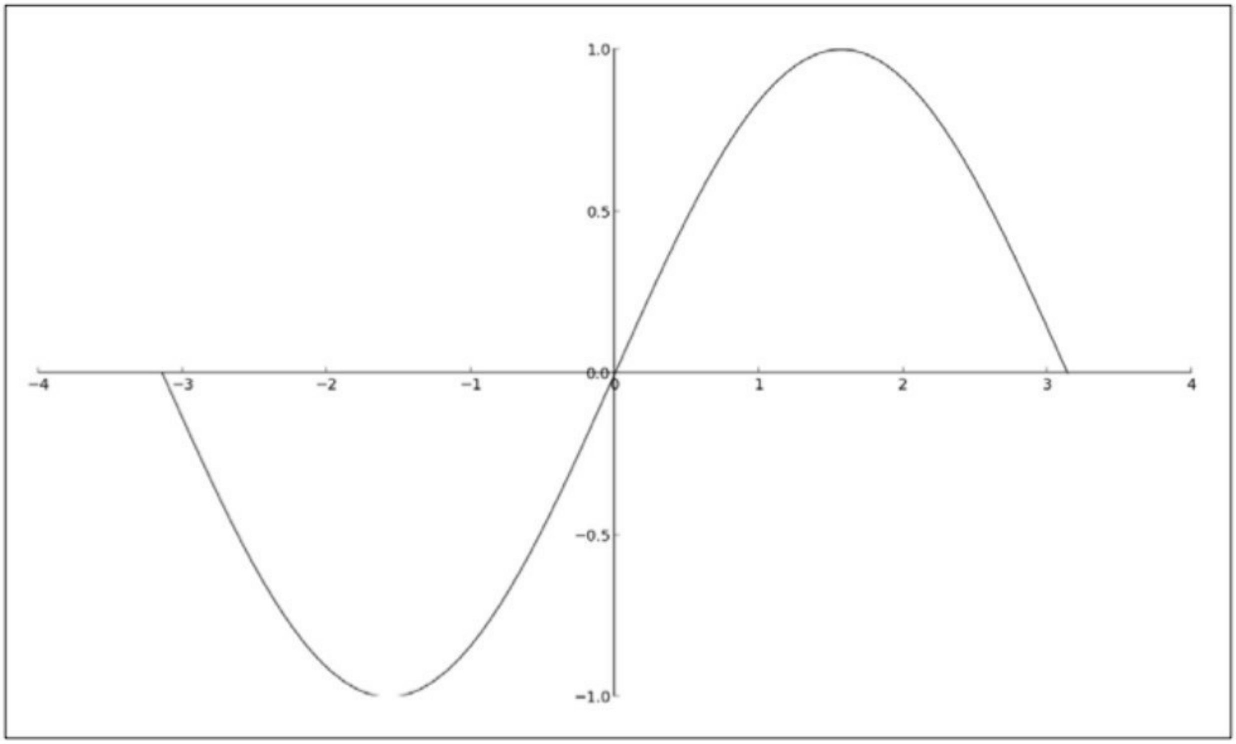


图3-11

3.8.2 工作原理

这段代码是取决于所绘制的图形的。我们把轴线移到位置（0，0），绘制了一个正弦函数曲线。（0，0）是图形的中心。

尽管如此，这段代码说明了如何把轴线移动到一个特定位置，以及怎样去掉不想显示的轴线。

3.8.3 补充说明

另外，轴线可以被限制在数据结束的地方结束，例如调用 set_smart_bounds (True)。在这种情况下，matplotlib 会尝试以一种复杂的方式设置边界，例如处理颠倒的界限，或者在数据延伸出视图的情况下裁剪线条以适应视图。

3.9 绘制直方图

直方图非常简单，但重要的是用它来显示正确的数据。目前我们仅涉及2D直方图。

直方图被用于可视化数据的分布估计。通常，在谈论直方图时我们会使用一些术语。表示一定间隔下数据点频率的垂直矩形称为bin。bin以固定的间隔创建，因此直方图的总面积等于数据点的数量。

直方图可以显示数据的相对频率，而不是使用数据的绝对值。在这种情况下，总面积就等于1。

直方图经常被用在图像处理软件中，作为可视化图像属性（如给定颜色通道上光的分布）的一种方式。这些图像直方图进一步可以应用在计算机视觉算法来检测峰值，用来辅助进行边缘检测、图像分割等。

在第5章“3D可视化”中，会有几小节来介绍3D直方图。

3.9.1 准备工作

我们想得到正确的bin数量，但是因为没有严格的规则来说明什么是最优bin数量，所以很难做到这一点。在怎样计算bin数量上有几种不同的理论，最简单的一个是基于上取整（ceiling）函数，这时 bins(k)等于 ceiling (max(x) – min(x)/h)，其中 x是绘制的数据集合，h为期望的bin宽度。这只是一种选项，因为正确显示数据的bin数量取决于真实的数据分布。

3.9.2 操作步骤

如果调用 matplotlib.pyploy.hist()来创建直方图，我们需要传入一些

参数，下面是一些最重要的参数。

◆ bins：可以是一个bin数量的整数值，也可以是表示bin的一个序列。默认值为10。

◆ range：bin 的范围，当 bins 参数为序列时，此参数无效。范围外的值将被忽略掉，默认值为None。

◆ normed：如果值为 True，直方图的值将进行归一化（normalized）处理，形成概率密度。默认值为False。

◆ histtype：默认为 bar类型的直方图。其他选项有以下几个。

● barstacked：用于多种数据的堆叠直方图。

● step：创建未填充的线形图。

● stepfilled：创建默认填充的线形图。histtype的默认值为 bar。

◆ align：用于bin边界之间矩形条的居中设置。默认值为mid，其他值为left和right。

◆ color：指定直方图的颜色。可以是单一颜色值或者颜色的序列。如果指定了多个数据集合，颜色序列将会设置为相同的顺序。如果未指定，将会使用一个默认的线条颜色。

◆ orientation：通过设置 orientation为 horizontal创建水平直方图。默认值为vertical。

下面的代码演示了hist()的用法。

```
import numpy as np
import matplotlib.pyplot as plt
mu = 100
sigma = 15
x = np.random.normal(mu, sigma, 10000)
ax = plt.gca()
# the histogram of the data
ax.hist(x, bins=35, color='r')
```

```
ax.set_xlabel('Values')
ax.set_ylabel('Frequency')
ax.set_title(r'$\mathrm{Histogram:}\ \mu=%d,\ \sigma=%d$' % (mu, sigma))
plt.show()
```

以上代码为数据样本创建了一个简洁的红色直方图，如图3-12所示。

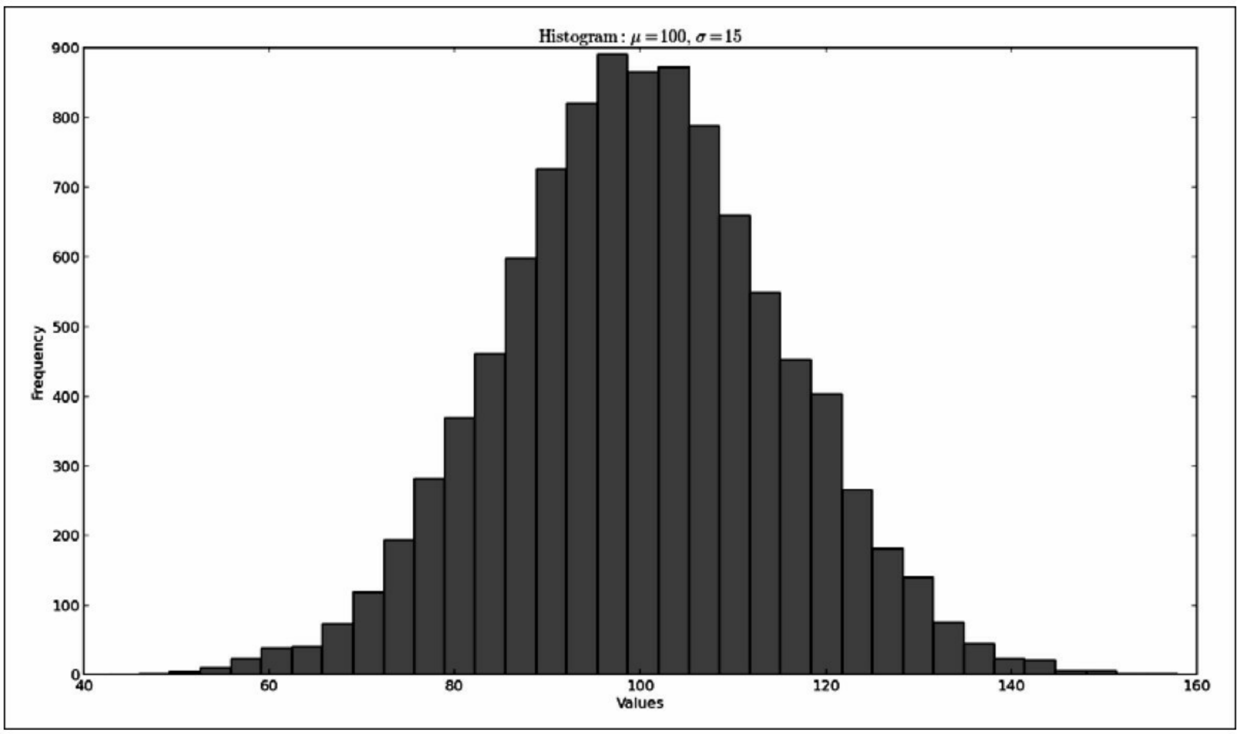


图3-12

3.9.3 工作原理

先生成一些正态分布数据，然后为直方图指定bin的数量为35，通过设置normed为True（或1）进行归一化处理，最后设置color为red(r)。

接下来，为图形添加标签和标题。这里我们利用 matplotlib 对LaTex 表达式的支持，在Python格式化字符中加入了数学符号。

3.10 绘制误差条形图

本节将展示如何创建柱状图以及如何绘制误差条。

3.10.1 准备工作

可以用误差条来可视化数据集合中的测量不确定度（uncertainty of measurement）或者指出错误。误差条可以很容易地表示误差偏离数据集合的情况。它们可以显示一个标准差（standard deviation）、一个标准误差（standard error）或者95%的置信区间（confidence interval）。因为在表示上没有统一标准，所以总是需要显式地表明误差条显示的是哪一种值（误差）。实验科学（experimental sciences）领域的大多数论文都应该在描述数据精度的时候包含误差条。

3.10.2 操作步骤

虽然只有两个必选参数——left和height，但是，我们经常会需要使用其他的参数。介绍如下。

◆ width：给定误差条的宽度，默认值是 0.8。

◆ bottom：如果指定了 bottom，其值会加到高度中，默认值为None。

◆ edgecolor：给定误差条边界颜色。

◆ ecolor：指定误差条的颜色。

◆ linewidth：误差条边界宽度，可以设为 None（默认值）和 0（此时误差条边界将不显示出来）。

◆ orientation：有 vertical和 horizontal两个值。

◆ xerr和 yerr：用于在柱状图上生成误差条。

一些可选参数（color、edgecolor、linewidth、xerr和yerr）可以是单一值，也可以是和误差条数目相同长度的序列。

3.10.3 工作原理

让我们用一个例子来说明误差条形图的绘制。

```
import numpy as np
import matplotlib.pyplot as plt
# generate number of measurements
x = np.arange(0, 10, 1)
# values computed from "measured"
y = np.log(x)
# add some error samples from standard normal distribution
xe = 0.1 * np.abs(np.random.randn(len(y)))
# draw and show errorbar
plt.bar(x, y, yerr=xe, width=0.4, align='center', ecolor='r',
color='cyan', label='experiment #1');
# give some explainations
plt.xlabel('# measurement')
plt.ylabel('Measured values')
plt.title('Measurements')
plt.legend(loc='upper left')
plt.show()
```

上述代码生成如图3-13所示的图形。

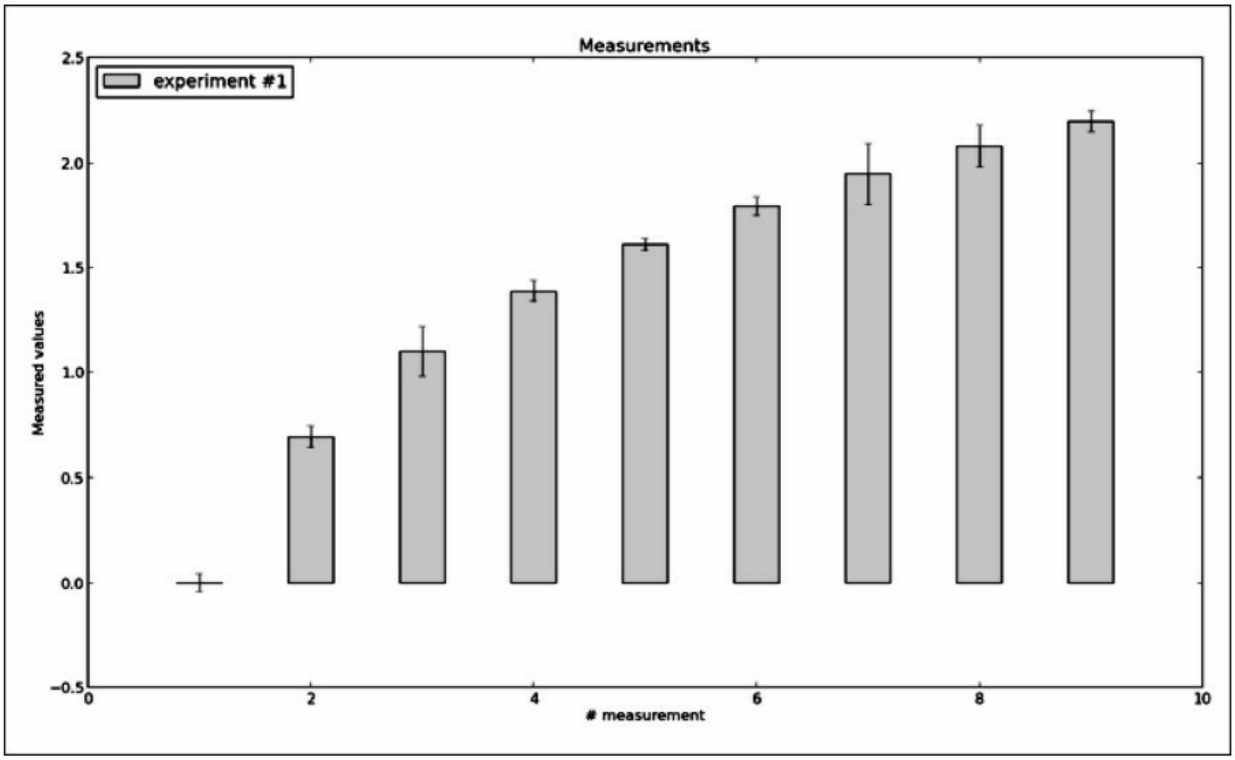


图3-13

为了绘制误差条，需要有一些度量值(x)；对于每一个度量值计算出的值(y)，我们得出了误差(xe)。

这里，我们用NumPy库来生成和计算值。标准分布已经能很好地满足演示的需要了，但是如果正好预先知道你的数据分布，可以做一些可视化原型来尝试一下不同的视图布局，以便找到展示信息的最佳选择。

如果正在准备为一个黑白版的媒介做可视化，另一个有意思的选择是使用阴影线（hatch）。阴影线的值如表3-4所示。

表3-4

阴影线的值	描述	阴影线的值	描述
/	斜线	x	交叉线
\	反斜线	o	小圆圈
|	垂直线	0	大圆圈
-	水平线	.	点
+	十字线	*	星号

3.10.4 补充说明

上文刚用到的误差条叫作对称误差条。如果数据集合的性质是误差在两个方向上（正向和负向）不同，也可以用非对称误差条来表示。

非对称误差条必须用一个两个元素的列表（比如一个二维数组）来指定xerr和yerr，其中第一个列表包含负向误差的值，第二个包含正向误差的值。

3.11 绘制饼图

饼图在很多方面很特别，最重要的一点是它显示的数据集合加起来必须等于100%，否则它就是无意义的、无效的。

3.11.1 准备工作

饼图描述数值的比例关系，其中每个扇区的弧长大小为其所表示的数量的比例。

饼图很紧凑，看上去很有美感，但是它们也因为难以对数量进行比较而备受批评。饼图的另一个不好的特征是它以特定角度（视角）的方式和一定颜色的扇形展示数据，这会使我们的感觉有倾向性，从而影响我们对于所呈现数据得出的结论。

下面演示用饼图呈现数据的不同方式。

3.11.2 操作步骤

首先，创建一个所谓的分裂式饼图（exploded pie chart）。

```
from pylab import *
# make a square figure and axes
figure(1, figsize=(6,6))
ax = axes([0.1, 0.1, 0.8, 0.8])
# the slices will be ordered
# and plotted counter-clockwise.
labels = 'Spring', 'Summer', 'Autumn', 'Winter'
# fractions are either x/sum(x) or x if sum(x) <= 1
```

```
x = [15, 30, 45, 10]
# explode must be len(x) sequence or None
explode=(0.1, 0.1, 0.1, 0.1)
pie(x, explode=explode, labels=labels,
autopct='%1.1f%%', startangle=67)
title('Rainy days by season')
show()
```

饼图如果绘制在一个正方形的图表中并且有正方形的坐标轴，看上去会非常漂亮。

饼图的每部分定义为 x/sum(x)，或者 x if sum(x) <= 1。通过给定一个分裂序列，可以获得分裂的效果，其中每一个元素表示每个圆弧间偏移量，为半径的百分比。用autopct参数来格式化绘制在圆弧中的标签，标签可以是一个格式化字符串或者是一个可调用的对象（函数）。

我们也可以使用一个布尔值的阴影参数给饼图添加阴影效果。

如果没有指定 startangle，扇区将从 x 轴（角度 0）开始逆时针排列；如果指定atartangle的值为90，饼图将从y轴开始。

绘制出的饼图如图3-14所示。

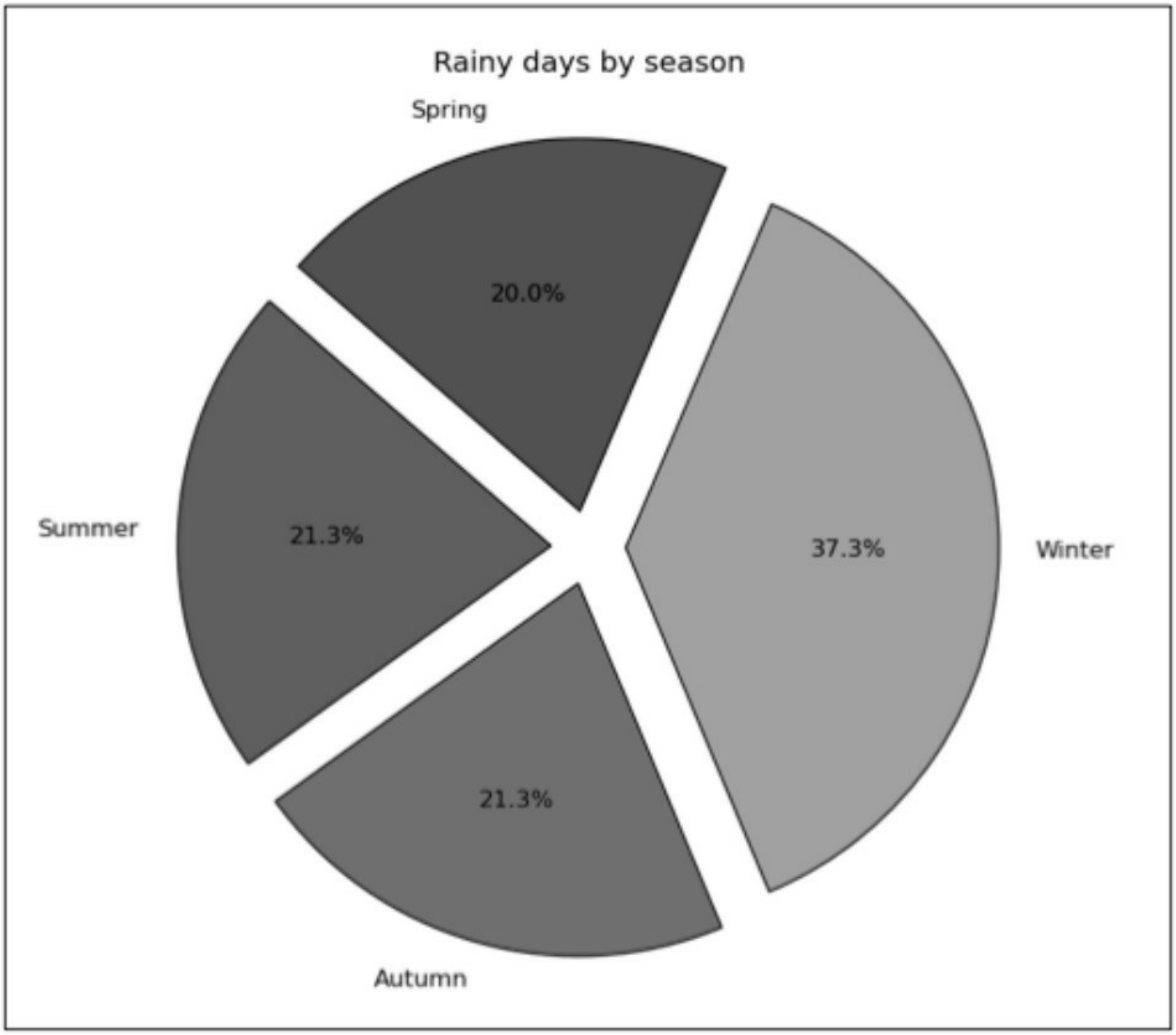

Rainy days by season
Spring
20.0%
Summer
21.3%
37.3%
Winter
21.3%
Autumn

图3-14

3.12 绘制带填充区域的图表

本节将展示如何对曲线下面的区域或者两个曲线之间的区域进行填充。

3.12.1 准备工作

matplotlib库允许我们对曲线间或者曲线下面的区域填充颜色，这样就可以向读者显示那部分区域的值。有些时候，这对读者（观察者）理解给定的特定信息是非常有必要的。

3.12.2 操作步骤

下面是一个关于如何填充两个轮廓线之间的区域的例子。

```
from matplotlib.pyplot import figure, show, gca
import numpy as np
x = np.arange(0.0, 2, 0.01)
# two different signals are measured
y1 = np.sin(2*np.pi*x)
y2 = 1.2*np.sin(4*np.pi*x)
fig = figure()
ax = gca()
# plot and
# fill between y1 and y2 where a logical condition is met
ax.plot(x, y1, x, y2, color='black')
ax.fill_between(x, y1, y2, where=y2>=y1, facecolor='darkblue',
```

```
interpolate= True)
    ax.fill_between(x, y1, y2, where=y2<=y1, facecolor='deeppink',
interpolate= True)
    ax.set_title('filled between')
    show()
```

3.12.3 工作原理

在生成预定义间隔的随机信号之后，用常规的plot()方法绘制出这两个信号的图形。然后，调用fill_between()并传入所需的必选参数。

如图 3-15 所示，fill_between()方法使用 x 为定位点选取 y 值（y1, y2），然后用几种预定义的颜色绘制出多边形。

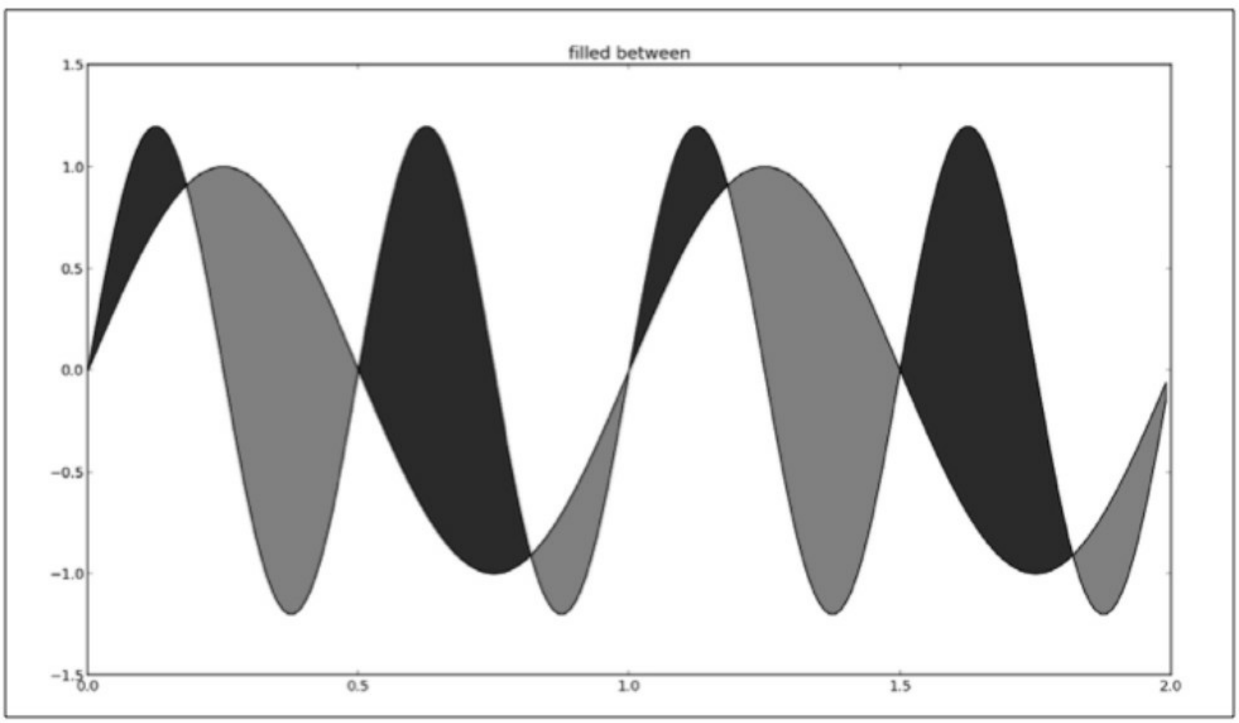


图3-15

用where参数指定一个条件来填充曲线，where参数接受布尔值（可以是表达式），这样就只会填充满足where条件的区域。

3.12.4 补充说明

像许多其他用于绘图的函数一样，fill_between()方法也接收许多参数，比如hatch （指定填充的样式替代颜色）和线条选项（linewidth和linestyle）。

另外一个方法是 fill_betweenx()，该方法有相似的填充特性，但是它是针对水平曲线的。

更通用的fill()方法提供了对任意多边形填充颜色或者阴影线的功能。

3.13 绘制带彩色标记的散点图

如果有两个变量，并且想标记出它们之间的相关关系（correlation），散点图是一种解决方案。

这种类型的图形也非常有用，它可以作为更高级的多维数据可视化的基础，比如绘制散点图矩阵（scatter plot matrix）。

3.13.1 准备工作

散点图显示两组数据的值。数据可视化的工作由一组并不由线条连接的点完成。每个点的坐标位置由变量的值决定。一个变量是自变量（或称为无关变量，independent variable），另一个是应变量（或称为相关变量，dependent variable）。应变量通常绘制在 y 轴上。

3.13.2 操作步骤

下述代码绘制了两幅图，一个是不相关数据，另一个是强正相关数据（strong positive correlation）。

```
import matplotlib.pyplot as plt
import numpy as np
# generate x values
x = np.random.randn(1000)
# random measurements, no correlation
y1 = np.random.randn(len(x))
# strong correlation
y2 = 1.2 + np.exp(x)
```

```
ax1 = plt.subplot(121)
plt.scatter(x, y1, color='indigo', alpha=0.3, edgecolors='white',
label='no correl')
plt.xlabel('no correlation')
plt.grid(True)
plt.legend()
ax2 = plt.subplot(122, sharey=ax1, sharex=ax1)
plt.scatter(x, y2, color='green', alpha=0.3, edgecolors='grey',
label='correl')
plt.xlabel('strong correlation')
plt.grid(True)
plt.legend()
plt.show()
```

在这里，我们也使用了很多参数，如用来设置图形颜色的color、用来设置点状标记（默认是circle）的marker、alpha（alpha透明度）、edgecolors（标记的边界颜色）和label（用于图例框）。

得到如图3-16所示的图形。

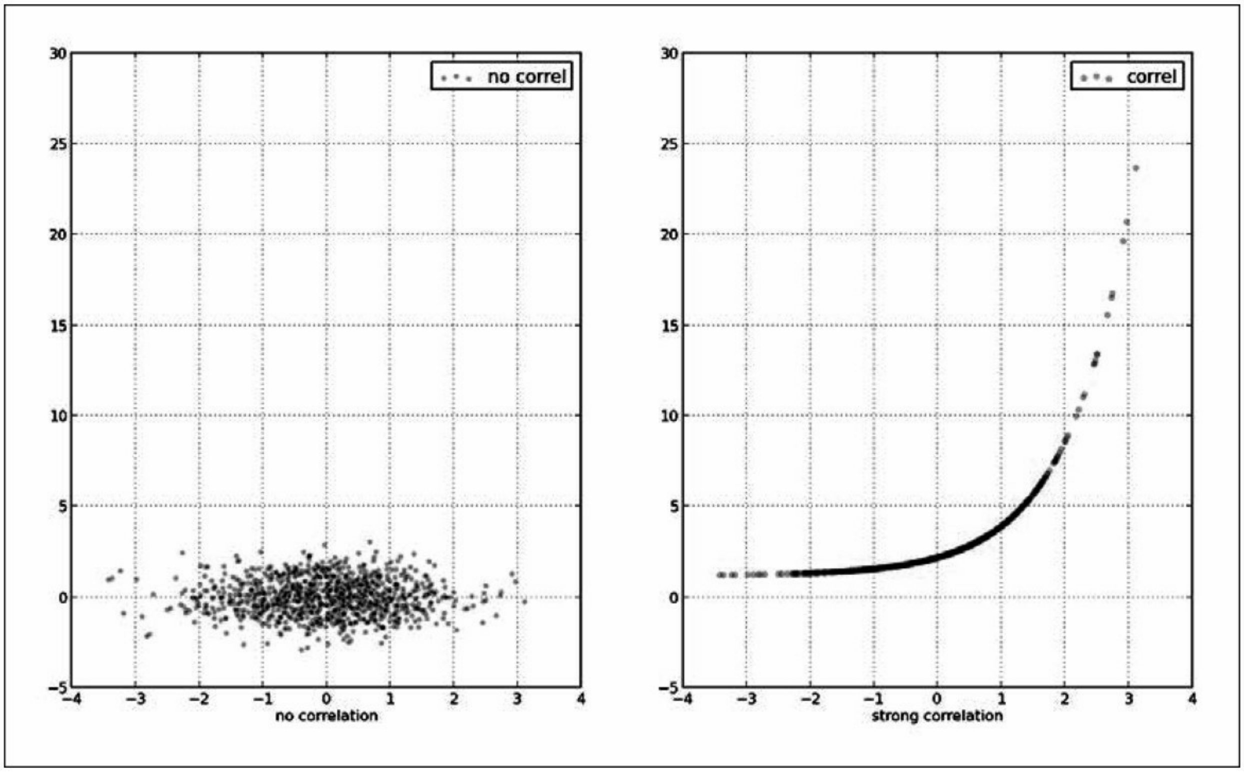


图3-16

3.13.3 工作原理

散点图通常在应用拟合回归函数之前绘制，用来识别两个变量间的关联。它很好地呈现了相关性的视觉画面，尤其是对于非线性关系的数据。matplotlib提供的scatter()函数用来绘制与 x 相同长度的一维数组（unidimensional array）y 的散点图。

注释

[1].在 Mac OS X 上调用 ishold()方法，返回为 True，和作者描述的不相符。

[2].原文为 Pi，应为作者笔误。

[3]. 原文为 matplotlib.pyploy.autoscale()，应为作者笔误。

[4].原文为 axspan，应为作者笔误。

[5]. 本书中 subplots 翻译为子区（和《Python 科学计算》中保持一致），其他的名词翻译遵循：figure 为图形或图表，axes 为坐标轴。

[6]. annotate 方法的第二个参数。

第4章 学习更多图表和定制化

在本章中，我们将学习以下内容。

◆ 设置坐标轴标签的透明度和大小

◆ 为图表线条添加阴影

◆ 向图表添加数据表

◆ 使用 subplots（子区）

◆ 定制化网格

◆ 创建等高线图

◆ 填充图表底层区域

◆ 绘制极线图

◆ 使用极坐标图可视化文件系统树

4.1 简介

在本章中，我们会研究matplotlib库的一些更高级的特性。我们将介绍更多的技术，来看看如何得到满意的可视化效果。

有时简单的图表不能够充分展现数据，本章我们会寻求数据展现中的一些重要问题的解决方案。我们将尝试使用多种类型的图表，或者创建不同图表的混合体来满足一些高级数据结构和特定的展现需求。

4.2 设置坐标轴标签的透明度和大小

Axes标签对于读者理解图表非常重要，它描述了图表中展现的数据内容。通过向axes添加标签，我们能够帮助读者更准确地理解图表所表达的信息。

4.2.1 准备工作

在深入分析代码之前，重要的是先了解matplotlib是如何组织图表的。

最上层是一个 Figure实例，包含了所有可见的和其他一些不可见的内容。该 Figure实例包含了一个Axes实例字段Figure.axes。Axes实例几乎包含了我们所关心的所有东西，如所有的线、点、刻度和标签。因此，当调用plot()方法时，就会向Axes.lines列表添加一个线条的实例（matplotlib.lines.Line2D）。如果绘制了一个直方图（通过调用hist()），就会向Axes.patches列表添加许多矩形（“patches[1]”是从MATLAB™继承来的一个术语，表示“颜色补片”的概念）。

Axes实例也包含了XAxis和YAxis实例的引用，分别指向相应的x轴和y轴。XAxis和YAxis管理坐标轴、标签、刻度、刻度标签、定位器和格式器的绘制，我们可以通过Axes.xaxis和Axes.yaxis分别引用它们。其实不必按照前面所说的方式通过XAxis或YAxis实例得到标签对象，因为matplotlib提供了一个helper方法（实际上是一个捷径）来迭代这些标签，它们是matplotlib.pyplot.xlabel()和matplotlib. pyplot.ylabel()。

4.2.2 操作步骤

我们现在将要创建一个新的图表，然后在其上面进行如下操作。

1.创建一个基于一些随机生成的数据的plot。

2.添加title和axes标签。

3.添加alpha设置。

4.向title和axes标签添加阴影效果。

```
import matplotlib.pyplot as plt
from matplotlib import patheffects
import numpy as np
data = np.random.randn(70)
fontsize = 18
plt.plot(data)
title = "This is figure title"
x_label = "This is x axis label"
y_label = "This is y axis label"
title_text_obj = plt.title(title, fontsize=fontsize,
verticalalignment='bottom')
title_text_obj.set_path_effects([patheffects.
withSimplePatchShadow()])
# offset_xy -- set the 'angle' of the shadow
# shadow_rgbFace -- set the color of the shadow
# patch_alpha -- setup the transparency of the shadow
offset_xy = (1, -1)
rgbRed = (1.0,0.0,0.0)
alpha = 0.8
# customize shadow properties
pe = patheffects.withSimplePatchShadow(offset_xy = offset_xy,
   shadow_rgbFace = rgbRed,
```

```
    patch_alpha = alpha)
# apply them to the xaxis and yaxis labels
xlabel_obj = plt.xlabel(x_label, fontsize=fontsize, alpha=0.5)
xlabel_obj.set_path_effects([pe])
ylabel_obj = plt.ylabel(y_label, fontsize=fontsize, alpha=0.5)
ylabel_obj.set_path_effects([pe])
plt.show()
```

4.2.3 工作原理

我们已经知道了所有熟悉的imports、生成数据的代码部分和基本的绘图技术，因此我们会省略它们。如果你不懂示例代码的前几行，请参考第2章的“了解你的数据”一节和第3章的“绘制图形并定制化”小节。

在绘制完数据集合的图表后，接下来准备添加标题和标签，并定制化它们的外观。

首先，添加一个标题。然后设置标题字体的大小，并设置标题文本的垂直对齐方式为bottom。如果不带参数地调用matplotlib.patheffects.withSimple PatchShadow()，会为标题添加默认的阴影效果。参数的默认值为offset_xy=(2,-2)，shadow_rgbFace=None和patch_alpha=0.7。标题文本的垂直对齐方式还有center、top和baseline，这里因为要为文本添加阴影，所以选择bottom。下一行代码为标题添加了阴影效果。路径效果（path effects）是matplotlib的matplotlib.patheffects模块的部分功能，支持matplotlib.text.Text和matplotlib.patches.Patch。

接着为x轴和y轴添加不同的阴影设置。首先，我们自定义相对于父对象[2]的阴影的位置（offset，偏移），然后设置阴影的颜色。颜色在这里表示为一个0.0～1.0之间浮点数的三元组，每一个浮点数代表一个

RGB通道。因此，红色表示为（1.0，0.0，0.0）（全红，无绿色，无蓝色）。

透明度（或者alpha）被设置为一个归一化的值，我们也想为其设置一个不同于默认值的值。

设置完毕后，实例化 matplotlib.patheffects.withSimplePatchShadow 对象,并将其引用保存在pe变量中以供后面的代码重用它。

为了能应用阴影效果，我们需要得到label对象。这再简单不过了，因为matplotlib. pyplot.xlabel()返回了该对象（matplotlib.text.Text）的引用。然后用它来调用set_path_effects([pe])方法。

最后把图表显示出来，真为我们做的工作感到骄傲。

4.2.4 补充说明

如果你不满足于 matplotlib.patheffects 目前提供的效果，可以继承 matplotlib. pathefffects._Base类，并重写draw_path方法。看看下面的代码和注释来了解一下是如何操作的。

https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/pathects.py#L47

4.3 为图表线条添加阴影

为了区分图表中的某一线条，或者仅仅为了保持包含图表在内的所有表格的总体风格，有时需要为图表线条（或者直方图）添加阴影效果。在本节中，我们将学习如何向图表添加阴影效果。

4.3.1 准备工作

为了向图表中的线条或者矩形条添加阴影，需要使用matplotlib内置的transformation框架，其位于matplotlib.transforms模块中。

为了理解所有这些是如何工作的，我们需要解释下matplotlib中的transformations框架是什么以及它们的工作原理。

Transformations 知道如何将给定的坐标从其坐标系转换到显示坐标系中，它们也知道如何将坐标从显示坐标系转换到它们自己的坐标系中。

表4-1总结了现有的坐标系以及它们描述的内容。

表4-1

坐标系	Transformation 对象	描　　述
Data	Axes.transData	表示用户的数据坐标系
Axes	Axes.transAxes	表示 Axes 坐标系，其中（0，0）表示轴的左下角，（1，1）表示轴的右上角
Figure	Figure.transFigure	是 Figure 坐标系，其中（0，0）表示图表的左下角，（1，1）表示图表的右上角
Display	None	表示用户视窗的像素坐标系，其中（0，0）表示视窗的左下角，（width，height）元组表示显示界面的右上角。这里的 width 和 height 都是以像素为单位的

注意，在Transformations对象列中，视窗坐标系是没有值的。这是

因为默认的坐标系就是Display坐标系，坐标总是在视窗坐标系下并以像素为单位。但这并没有太大的用处，因为大多数情况下我们想把坐标归一化到 Figure、Axes 或者一个 Data 坐标系中。

这个框架能让我们把现有对象转化成一个偏移对象，也就是说，把对象放置到偏离原来对象一段距离的地方。

4.3.2 操作步骤

下面是向图表添加阴影效果的代码。在下一小节中会对代码进行解释。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms
def setup(layout=None):
    assert layout is not None
    fig = plt.figure()
    ax = fig.add_subplot(layout)
    return fig, ax
def get_signal():
    t = np.arange(0., 2.5, 0.01)
    s = np.sin(5 * np.pi * t)
    return t, s
def plot_signal(t, s):
    line, = axes.plot(t, s, linewidth=5, color='magenta')
    return line,
def make_shadow(fig, axes, line, t, s):
    delta = 2 / 72. # how many points to move the shadow
```

```
    offset = transforms.ScaledTranslation(delta, -delta,
fig.dpi_scale_trans)
    offset_transform = axes.transData + offset
    # We plot the same data, but now using offset transform
    # zorder -- to render it below the line
    axes.plot(t, s, linewidth=5, color='gray',
        transform=offset_transform,
        zorder=0.5 * line.get_zorder())
if __name__ == "__main__":
    fig, axes = setup(111)
    t, s = get_signal()
    line, = plot_signal(t, s)
    make_shadow(fig, axes, line, t, s)
    axes.set_title('Shadow effect using an offset transform')
    plt.show()
```

4.3.3 工作原理

我们从后部分代码的 if__name__检查语句之后开始阅读。首先通过setup()创建figure和axes。然后，得到一个信号（或者说生成一个正弦波数据）。在plot_signal()方法中绘制出基本的信号图。最后，进行阴影坐标转换并在make_shadow()方法中绘制出阴影。

使用偏移效果创建一个偏移对象，把阴影放置在原始对象之下并偏移几个点的距离。

原始对象是一个简单的正弦波，用标准的plot()方法进行绘制。

matplotlib 包含一个 transformations helper——matplotlib.transforms.Scaled Translation——来添加偏移转换。

dx和dy的值以点为单位。因为点是1/72英寸，向右移动偏移对象2pt，向下移动偏移对象2pt。



如果想了解更多关于如何转换点为1/72[3]英寸的知识，请阅读一下这篇Wikipedia 文章，参见http://en.wikipedia.org/wiki/ Point_%28 typography%29。

可以使用matplotlib.transforms.ScaledTransformation(xtr, ytr, scaletr)方法。这里，xtr和ytr是转换的偏移量，scaletr是一个转换可调用对象（callable），在转换时和显示之前对xtr和ytr进行比例调整。其最常用的情况是从点转换到显示区域，如 DPI，这样偏移始终保持在相同的位置而与实际的输出设备无关（可以是显示器或者打印的材料）。我们使用的可调用对象已经内置在matplotlib中，可以从Figure.dpi_scale_trans得到。

然后，用这些转换把数据绘制出来。

4.3.4 补充说明

使用 transforms 添加阴影只是这个框架的一种但不是最流行的用法。为了用 trans formations框架做更多的事情，需要了解transformation管道工作原理的详细内容以及有哪些扩展点（继承以及如何继承哪些类）。这非常简单，因为matplotlib是开源的，即使一些代码没有很好的文档，你也可以阅读、使用源码或者做些修改，进而为matplotlib总体的质量和可用性做些贡献。

4.4 向图表添加数据表

虽然matplotlib主要是一个绘图的库，但它可以在绘图时帮我们做一些琐事，比如在漂亮的图表旁放置一个整齐的数据表格。本节将学习如何在图表中的图形旁显示一个数据表格。

4.4.1 准备工作

首先，重要的是理解为什么要向图表添加表格。为数据绘制可视化图形的主要目的是解释那些不能理解（或者很难理解）的数据值。现在，我们想把数据添加回来。仅仅在图表下面生硬地添加一张大表格显然是不明智的做法。

但是，通过精心挑选的，来自数据整体集合的总结性的或者突出强调的值可以识别出图表的重要部分，或者在一些地方强调一些非常重要的值。在这些地方，这些精确的值（例如以USD为单位的年销售额）是非常重要的（或者是必需的）。

4.4.2 操作步骤

这段代码向图表添加了一个示例表格。

```
import matplotlib.pylab as plt
import numpy as np
plt.figure()
ax = plt.gca()
y = np.random.randn(9)
col_labels = ['col1','col2','col3']
```

```
row_labels = ['row1','row2','row3']
table_vals = [[11, 12, 13], [21, 22, 23], [28, 29, 30]]
row_colors = ['red', 'gold', 'green']
my_table = plt.table(cellText=table_vals,
    colWidths=[0.1] * 3,
    rowLabels=row_labels,
    colLabels=col_labels,
    rowColours=row_colors,
    loc='upper right')
plt.plot(y)
plt.show()
```

上述代码段生成如图4-1所示的图表。

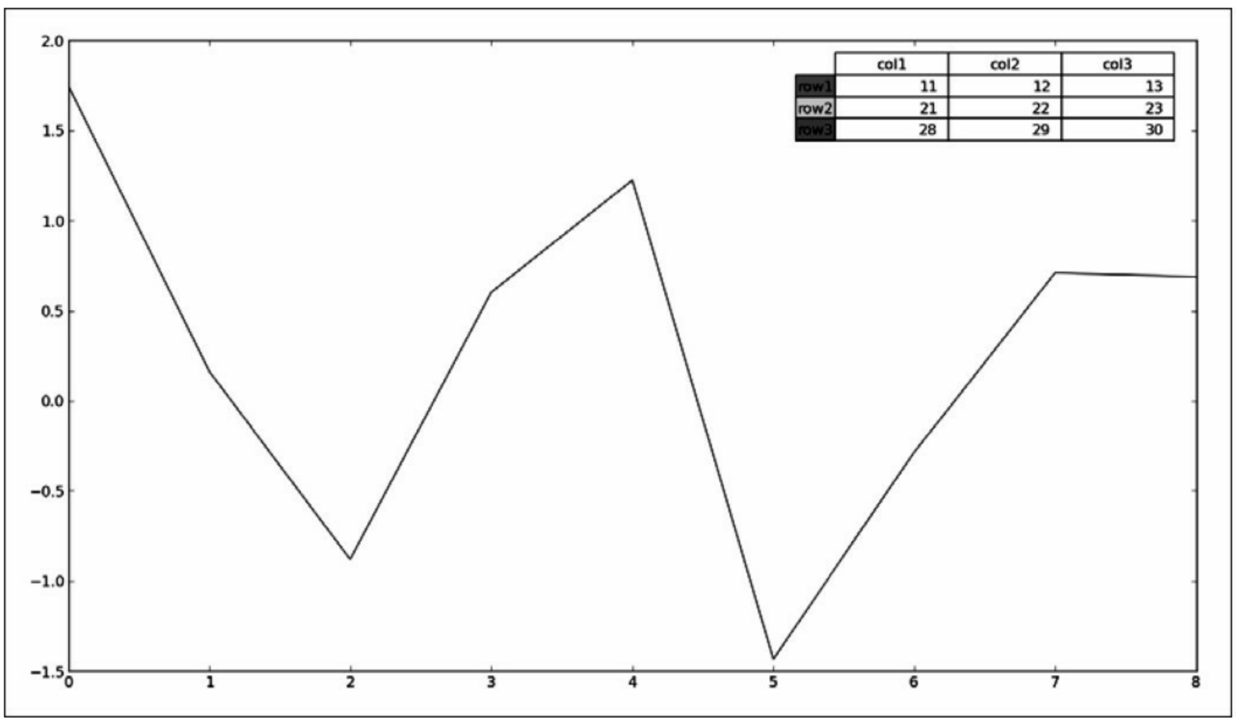


图4-1

4.4.3 工作原理

使用plt.table()方式创建了一个带单元格的表格，并把它添加到当前坐标轴中。表格可以有（可选的）行标题和列标题。每个单元格包含文本或补片。表格的列宽和行高是可以指定的。返回值是一个组成表格的对象（文本、线条和补片实例）的序列。

基本的函数签名如下。

```
table(cellText=None, cellColours=None,
   cellLoc='right', colWidths=None,
   rowLabels=None, rowColours=None, rowLoc='left',
   colLabels=None, colColours=None, colLoc='center',
   loc='bottom', bbox=None)
```

函数实例化并返回一个 matplotlib.table.Table实例。只有一种方式把表格添加到图表中，这也是 matplotlib 通常的情况。可以直接访问这个面向对象的接口。在用add_table()方法把图表添加到坐标轴实例之前，可以用 matplotlib. table.Table类直接对表格进行微调。

4.4.4 补充说明

如果直接创建一个matplotlib.table.Table类的实例，在把它添加到axes实例前你可以有更多的控制。可以使用 Axes.add_table(table)方法把table 实例添加到axes，这里的table是matplotlib.table.Table类的实例。

4.5 使用subplots(子区)

如果你是从开头阅读本书，一定对subplot类非常熟悉。subplot派生自axes，位于subplot实例的规则网格中。我们将要解释和演示如何以高级的方式使用子区。

本节将学习如何在plot中创建定制的子区配置项。

4.5.1 准备工作

子区的基类是 matplotlib.axes.SubplotBase。子区是 matplotlib.axes.Axes的实例，但提供了helper方法来生成和操作图表中的一系列Axes。

有一个matplotlib.figure.SubplotParams类，包括subplot的所有参数。尺寸是被归一化的图表的宽度或者高度。我们已经知道，如果不指定任何定制化的值，subplot将会从rc参数中读取参数值。

脚本层（matplotlib.pyplot）有操作子区的一些helper方法。

matplotlib.pyplot.subplots 用来方便地创建普通布局的子区。我们可以指定网格的大小——子区网格的行数和列数。

我们可以创建共享x或者y轴的子区，这通过使用sharex或者sharey关键字参数来完成。sharex参数可以设置为True，这样x轴就被所有的子区共享。这样一来，刻度标签只在最后一行的子区上可见。它们也可以被设置为字符串，枚举值如row、col、all或者none。值all和True相同，值none和False相同。如果设置为row，则每一个子区行共享x轴坐标；如果设置为col，则每一个子区列共享y轴坐标。matplotlib.pyplot. subplots方法返回一个（fig，ax）元组，其中ax可以是一个坐标轴实例；当创建

多个子区时，ax是一个坐标轴实例的数组。

我们用matplotlib.pyplot.subplots_adjust来调整子区的布局。关键字参数指定了图表中子区的坐标（left、right、 bottom和 top)，其值是归一化的图表大小的值。可以用 wspace和 hspace参数指定子区间空白区域的大小，参数值为相应宽度和高度的归一化值。

4.5.2 操作步骤

我们将演示matplotlib工具包中另一个helper函数——subplot2grid——的例子。我们定义了网格的几何形状和子区的位置。注意位置是基于0的，而不是像在plot.subplot()中那样基于1。也可以使用colspan和rowspan来让子区跨越给定网格中的多个行和列。例如，创建一个图表，通过subplot2grid添加不同的子区布局，并重新配置刻度标签大小。

显示图形代码如下。

```
import matplotlib.pyplot as plt
plt.figure(0)
axes1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
axes2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
axes3 = plt.subplot2grid((3, 3), (1, 2))
axes4 = plt.subplot2grid((3, 3), (2, 0))
axes5 = plt.subplot2grid((3, 3), (2, 1), colspan=2)
# tidy up tick labels size
all_axes = plt.gcf().axes
for ax in all_axes:
    for ticklabel in ax.get_xticklabels() + ax.get_yticklabels():
        ticklabel.set_fontsize(10)
plt.suptitle("Demo of subplot2grid")
```

```
plt.show()
```

当执行上述代码时，将创建出如图4-2所示的图形。

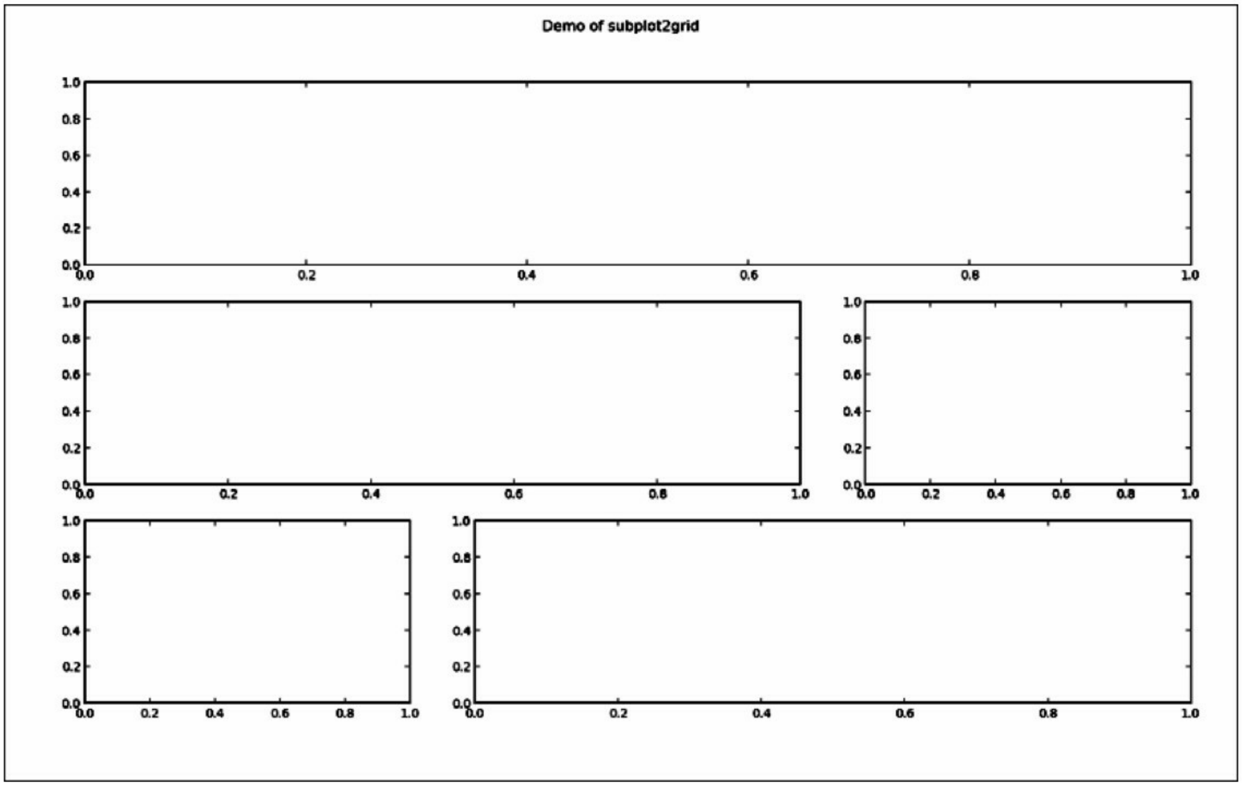


图4-2

4.5.3 工作原理

向subplot2grid方法传入形状参数、位置（loc）参数和可选的rowspan及colspan参数。这里一个重要的区别是位置从0开始索引，而figure.add_subplot从1开始索引。

4.5.4 补充说明

以下是一个以另一种方式定制化当前axes或者subplot的例子。

```
axes = fig.add_subplot(111)
rectangle = axes.patch
```

```
rectangle.set_facecolor('blue')
```

这里我们看到每一个axes实例包含了一个引用rectangle实例的patch字段，此字段代表当前axes实例的背景。我们可以更新该实例的属性，进而更新当前axes的背景。例如，可以改变其颜色，也可以加载一副图像以添加水印保护。

也可以先创建一个补片，然后把它添加到axes的背景上。

```
fig = plt.figure()
axes = fig.add_subplot(111)
rect = matplotlib.patches.Rectangle((1,1), width=6, height=12)
axes.add_patch(rect)
# we have to manually force a figure draw
axes.figure.canvas.draw()
```

4.6 定制化网格

在线条或者图表下面添加网格是非常有用的，它可以帮助肉眼识别出图案的不同，并且帮助我们比较图表中的图形。我们需要使用matplotlib.pyplot.grid来设置网格的可见度、密度和风格，或者是否显示网格。

本节将学习如何打开或关闭网格，以及如何改变网格上的主刻度和次刻度。

4.6.1 准备工作

最常用的网格定制化功能可以用 matplotlib.pyplot.grid helper 函数来完成。

为了看到其交互效果，在ipython–pylab下运行下面的代码。对plt.grid()的基本的调用将会在由 IPython PyLab 环境开启的当前交互式会话中切换网格的可见性，如图 4-3所示。

```
In [1]: plt.plot([1,2,3,3.5,4,4.3,3])
Out[1]: [<matplotlib.lines.Line2D at 0x3dcc810>]
```

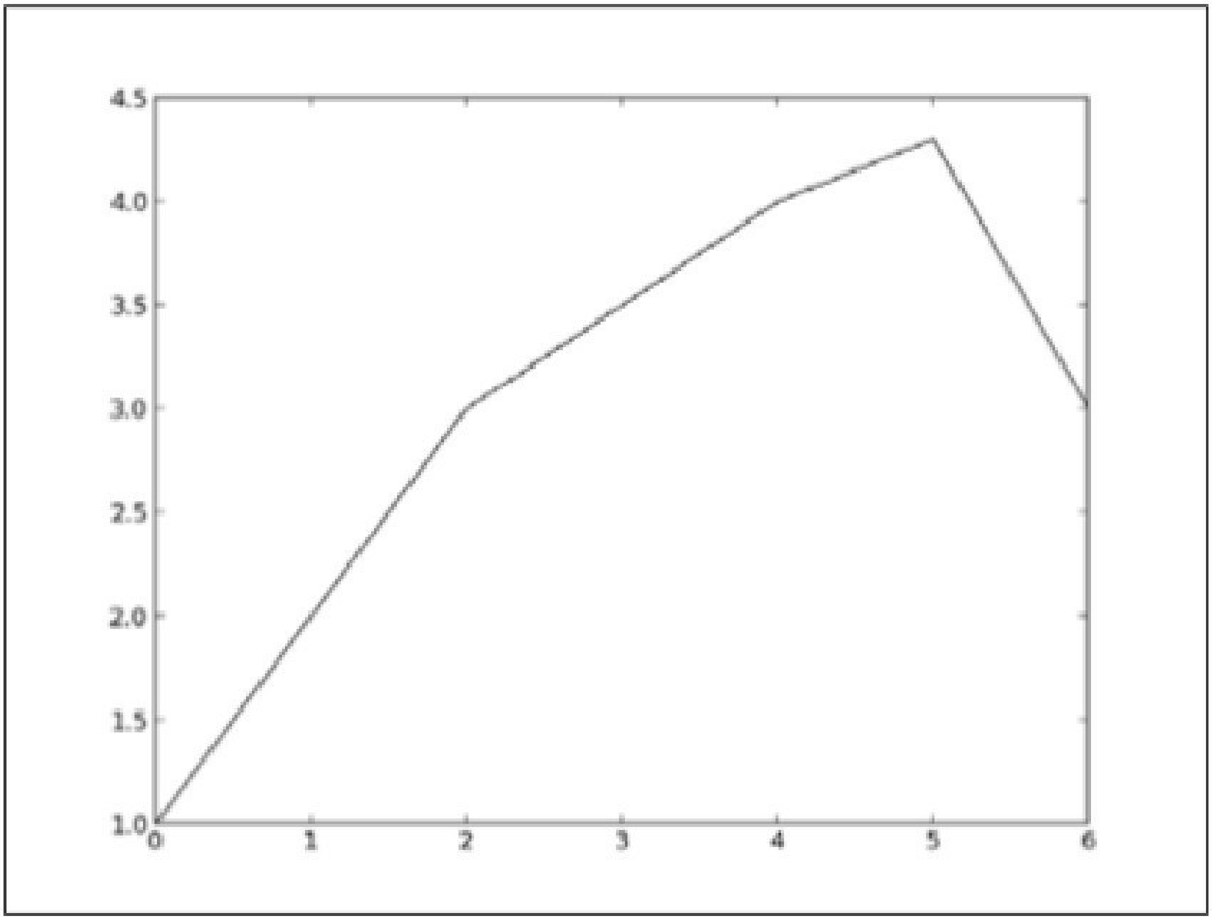


图4-3

现在我们可以在同一个图表中切换网格。

In [2]: plt.grid()

把网格打开，如图4-4所示。

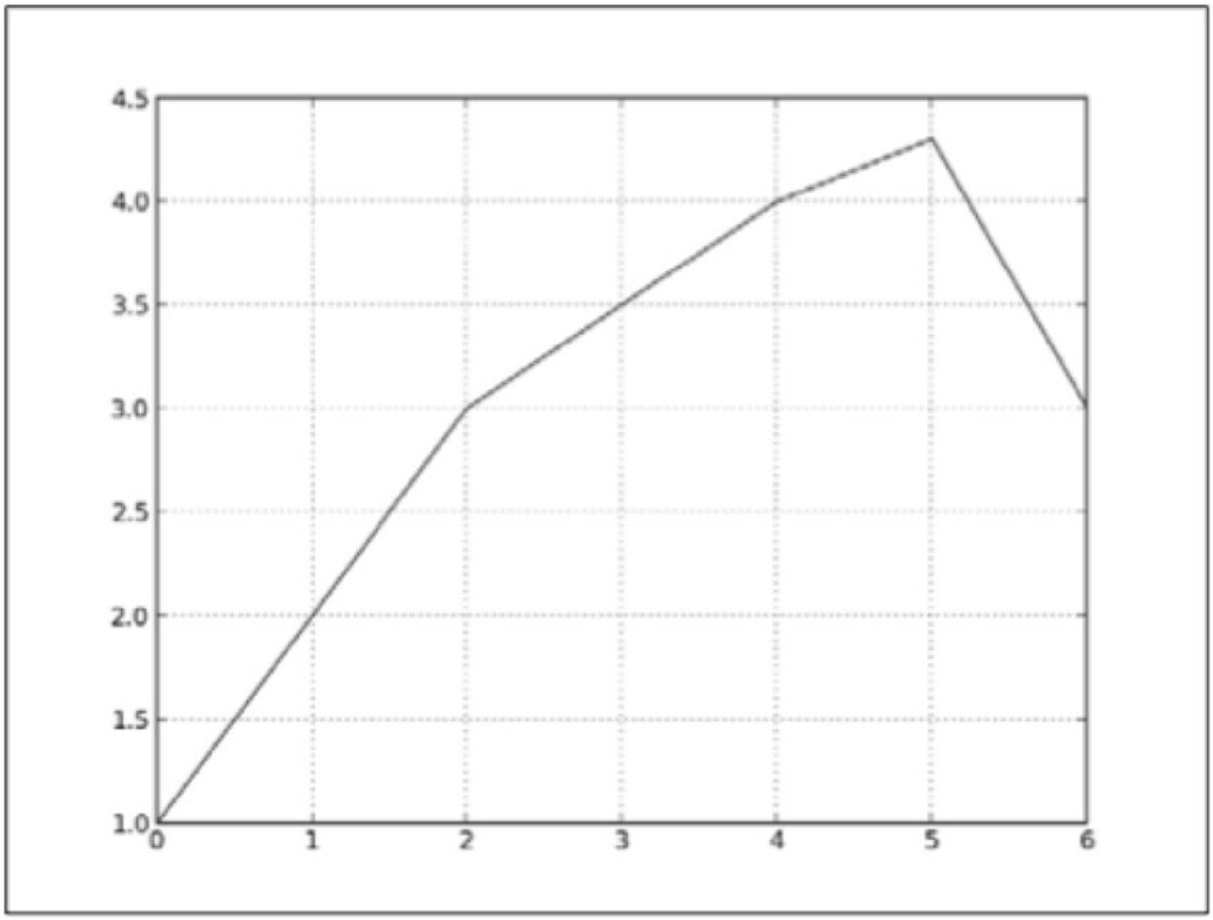


图4-4

然后关闭网格，如图4-5所示。

```
In [3]: plt.grid()
```

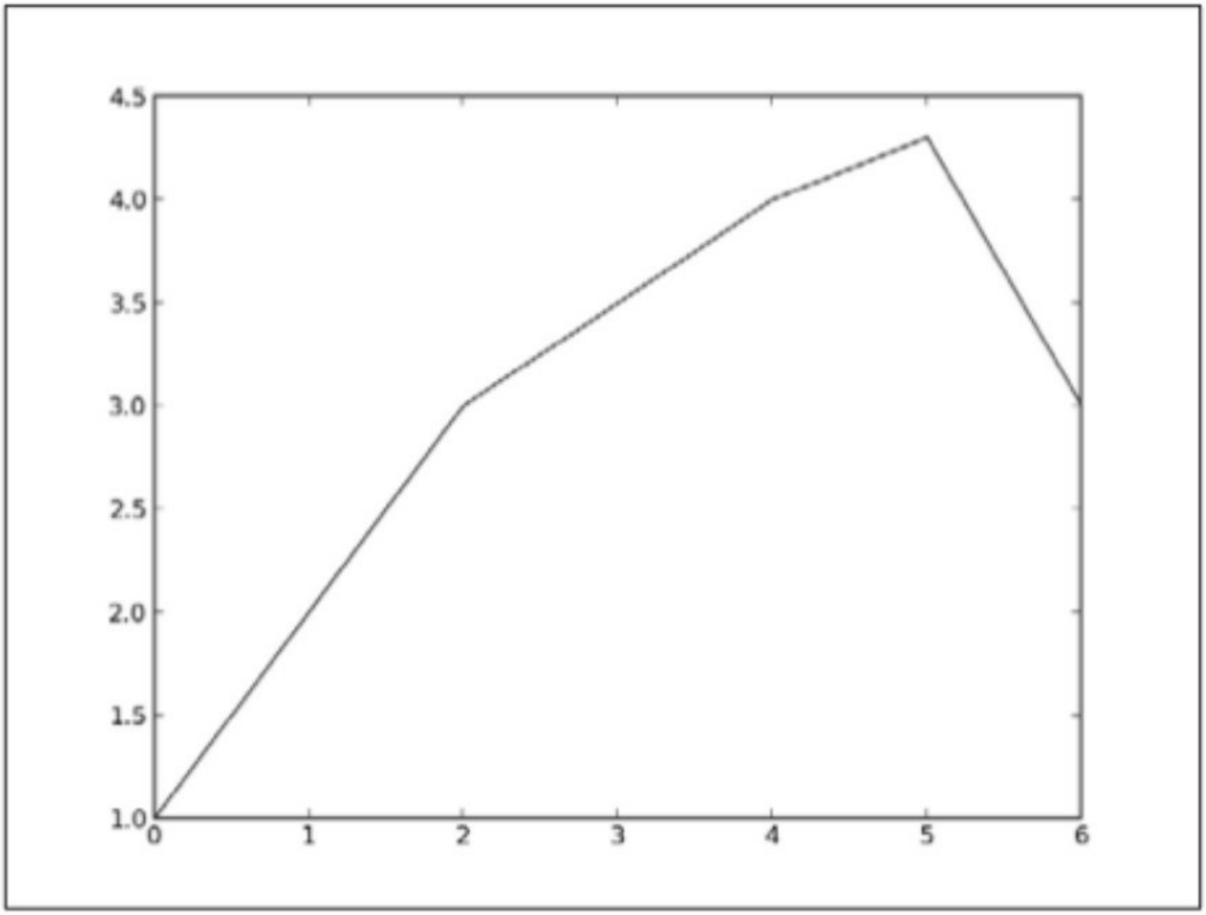


图4-5

除了只是打开或关闭网格之外，还能进一步定制化网格的外观。

我们可以仅通过主刻度或者次刻度，或者同时通过两个刻度来操作网格。因此，函数参数which可以是'major'、'minor'，或者'both'。与此类似，我们可以通过参数axis分别控制水平刻度和垂直刻度，参数值可以是'x'、'y'，或者'both'。

所有其他属性通过kwargs参数传入，代表一个matplotlib.lines.Line2D实例可以接受的标准属性集合，比如color、linestyle和linewidth。这里有一个例子。

```
ax.grid(color='g', linestyle='--', linewidth=1)
```

4.6.2 操作步骤

这非常不错，但是我们想要做更多的定制化。为此，我们需要深入地了解matplotlib和mpl_toolkits，并找到能以一个简单且可管理的方式创建坐标轴网格的AxesGrid模块。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from matplotlib.cbook import get_sample_data
def get_demo_image():
    f = get_sample_data("axes_grid/bivariate_normal.npy",
asfileobj=False)
    # z is a numpy array of 15x15
    Z = np.load(f)
    return Z, (-3, 4, -4, 3)
def get_grid(fig=None, layout=None, nrows_ncols=None):
    assert fig is not None
    assert layout is not None
    assert nrows_ncols is not None
    grid = ImageGrid(fig, layout, nrows_ncols=nrows_ncols,
            axes_pad=0.05, add_all=True, label_mode="L")
    return grid
def load_images_to_grid(grid, Z, *images):
    min, max = Z.min(), Z.max()
    for i, image in enumerate(images):
        axes = grid[i]
        axes.imshow(image, origin="lower", vmin=min, vmax=max,
            interpolation="nearest")
if __name__ == "__main__":
```

数据。

```
fig = plt.figure(1, (8, 6))
grid = get_grid(fig, 111, (1, 3))
Z, extent = get_demo_image()
# Slice image
image1 = Z
image2 = Z[:, :10]
image3 = Z[:, 10:]
load_images_to_grid(grid, Z, image1, image2, image3)
plt.draw()
plt.show()
```

上述代码绘制出如图4-6所示的图形。

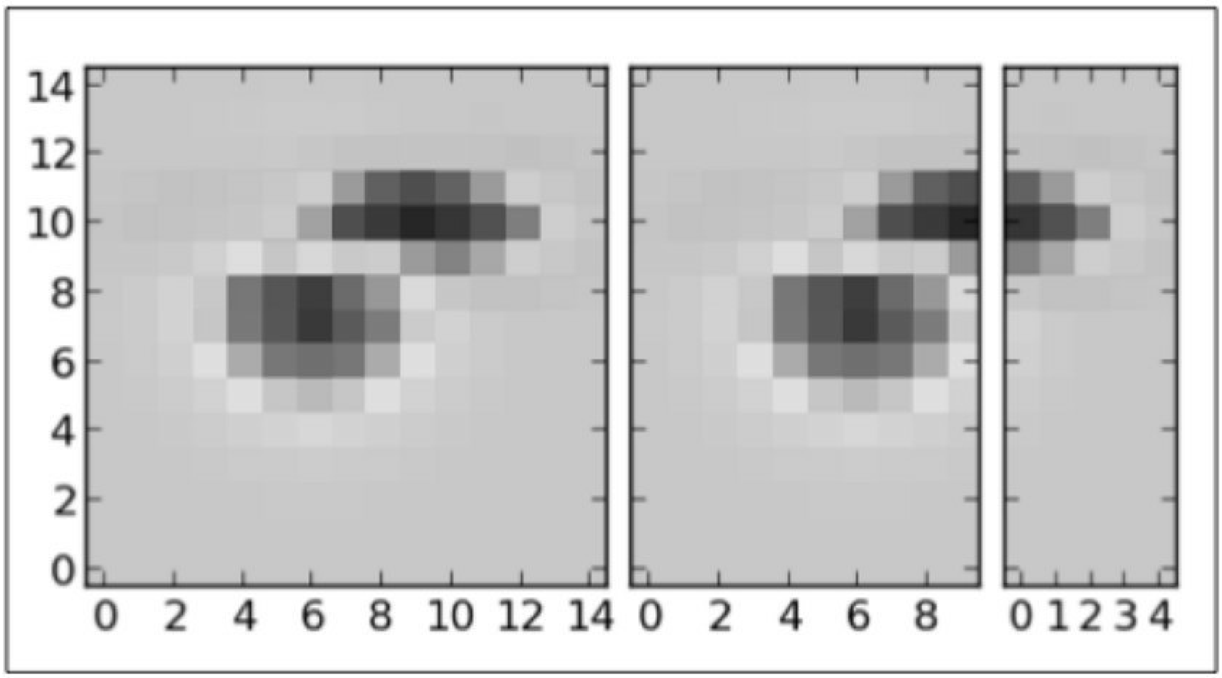


图4-6

4.6.3 工作原理

在函数get_demo_image中，我们从matplotlib的样本数据目录中加载数据。

grid列表保存了axes网格（此例中是ImageGrid）。

变量image1、image2、image3保存了Z的切片数据，这些数据是根据grid列表的多个坐标轴切分的。

循环遍历所有的网格，调用标准的imshow()方法绘制出image1、image2、image3[4]的数据。matplotlib确保所有图形的渲染是整洁的，排列是整齐的。

4.7 创建等高线图

等高线图（contour plot）显示的是矩阵的等值线（isolines）。等值线是用数值相等的各点连成的曲线。数值通过一个有两个参数的函数[5]获得。

本节将学习如何创建等高线图。

4.7.1 准备工作

Z矩阵的等高线图由许多等高线表示，这里的Z被视为相对于X-Y平面的高度。Z的最小值为2，并且必须包含至少两个不同的值。

等高线图的缺陷之一是如果在编码时不为等值线添加标签，它将毫无意义，因为我们不能分辨出最高点和最低点，或者找出局部极小值。

我们需要为等高线添加标签。可以使用标签（clabel()）或者colormaps为等值线添加标签。如果你的输出媒介允许使用颜色，colormaps 是首选，因为观察者将更容易理解数据。

等高线图的另一个风险是如何选择要绘制的等值线数量。如果选择的太多，图表就会变得太密集从而难以理解；如果选择的太少，将丢失信息，从而对数据做出不同的理解。

函数contour()会自动猜测出将绘制多少等值线，但我们也可以指定数量。

在matplotlib中，用matplotlib.pyplot.contour绘制等高线图。

这里有两个相似的函数：contour()绘制等高线，contourf()绘制填充的等高线。我们将只演示contour()，但是几乎所有内容对contourf()都是适用的。而且，它们的参数几乎相同。

contour()函数可以有不同的调用签名（如表4-2所示），这取决于我们拥有的数据和（或者）我们想可视化的属性。

表4-2

调用签名	描述
contour(Z)	绘制Z（数组）的等高线。自动选择水平值
contour(X, Y, Z)	绘制X、Y和Z的等高线。X和Y数组为(x, y)平面坐标(surface coordinates)
contour(Z, N) contour(X, Y, Z, N)	绘制Z的等高线，其中水平数由N决定。自动选择水平值
contour(Z, V) contour(X, Y, Z, V)	绘制等高线，水平值在V中指定
contour(…, V)	填充V序列中的水平值之间的len(V)-1个区域
contour(Z, **kwargs)	使用关键字参数控制一般线条属性（颜色、线宽、起点，颜色映射表（color map）等）

X、Y和Z的形状和维度存在一定的限制。例如，X和Y可以是二维的，与Z形状相同。如果它们是一维的，则X的长度等于Z的列数，Y的长度将等于Z的行数。

4.7.2 操作步骤

在下面的代码示例中，我们将进行以下操作。

1.实现一个方法来模拟信号处理器。

2.生成一些线性信号数据。

3.把数据转换到合适的矩阵中供矩阵操作使用。

4.绘制等高线。

5.添加等高线标签。

6.显示图形。

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
def process_signals(x, y):
```

```
    return (1 - (x ** 2 + y ** 2)) * np.exp(-y ** 3 / 3)
x = np.arange(-1.5, 1.5, 0.1)
y = np.arange(-1.5, 1.5, 0.1)
# Make grids of points
X, Y = np.meshgrid(x, y)
Z = process_signals(X, Y)
# Number of isolines
N = np.arange(-1, 1.5, 0.3)
# adding the Contour lines with labels
CS = plt.contour(Z, N, linewidths=2, cmap=mpl.cm.jet)
plt.clabel(CS, inline=True, fmt='%1.1f', fontsize=10)
plt.colorbar(CS)
plt.title('My function: $z=(1-x^2+y^2) e^{-(y^3)/3}$')
plt.show()
```

生成如图4-7所示的图表。

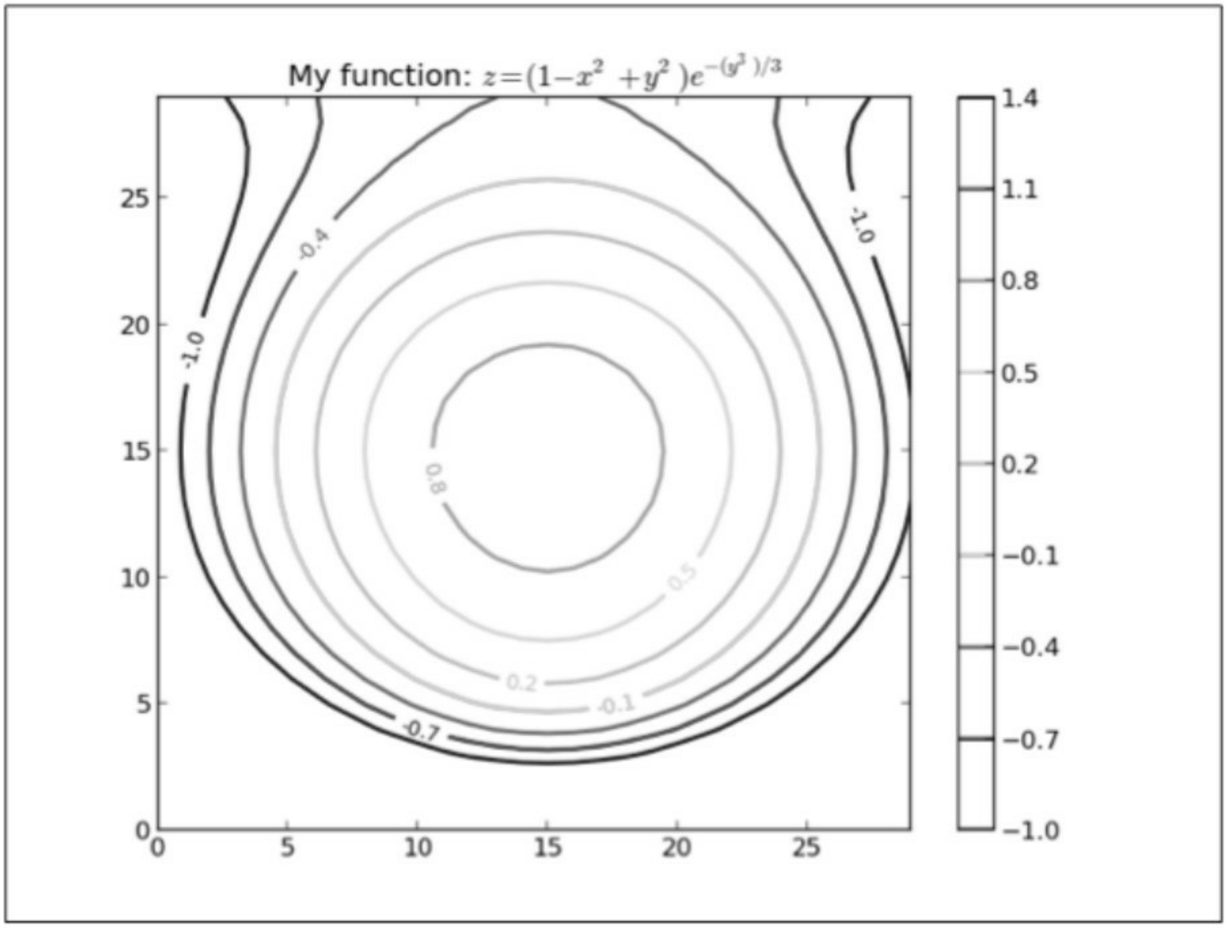


图4-7

4.7.3 工作原理

我们从 numpy借助少数几个 helper 方法来创建范围和矩阵。

在对my_function[6]求值并存储在Z之后，简单地调用contour，并传入Z和等值线水平数量。

此时，可以尝试用 N arange()调用中的第三个参数做个实验。例如，尝试将N=np.arange(-1, 1.5, 0.3)的参数做一些修改，将值 0.3改为 0.1或 1，来体验一下对相同数据进行不同编码时，它在等高线图中呈现的差异。

此外，我们通过简单地传入一个

CS（matplotlib.contour.QuadContourSet实例）向图表添加了一个颜色映射表。

4.8 填充图表底层区域

在matplotlib中绘制一个填充多边形的基本方式是使用matplotlib.pyplot.fill。

该方法接受和matplotlib.pyplot.plot相似的参数，即多个x、y对和其他Line2D属性。函数返回被添加的Patch实例的列表。

本节将学习如何为特定的图形交集区域填充阴影。

4.8.1 准备工作

除了如　histogram()等固有的绘制闭合的填充多边形的绘图函数之外，matplotlib 还提供了几个方法来帮助我们绘制填充的图形。

我们已经提到了一个——matplotlib.pyplot.fill，另外还有 matplotlib.pyplot.fill_between()和matplotlib.pyplot.fill_betweenx()[7]函数。这些方法填充两条曲线间的多边形。fill_between()和fill_betweenx()主要的区别是后者填充x轴的值之间的区域，而前者填充y轴的值之间的区域。

函数fill_between接收参数x（数据的x轴数组）和y1及y2（数据的y轴数组）。通过参数，我们可以指定条件来决定要填充的区域。这个条件是一个布尔条件，通常指定y轴值范围。默认值为None，表示填充所有区域。

4.8.2 操作步骤

从一个简单的例子开始，我们将填充一个简单函数下面的区域。

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from math import sqrt
t = range(1000)
y = [sqrt(i) for i in t]
plt.plot(t, y, color='red', lw=2)
plt.fill_between(t, y, color='silver')
plt.show()
```

上述代码生成如图4-8所示的图形。

它非常直观的让我们了解了 fill_between()是如何工作的。值得注意的是，fill_between()只是绘制了一个填充了颜色（'silver'）的多边形区域，所以我们需要绘制实际的函数线条，当然是使用plot（）了。

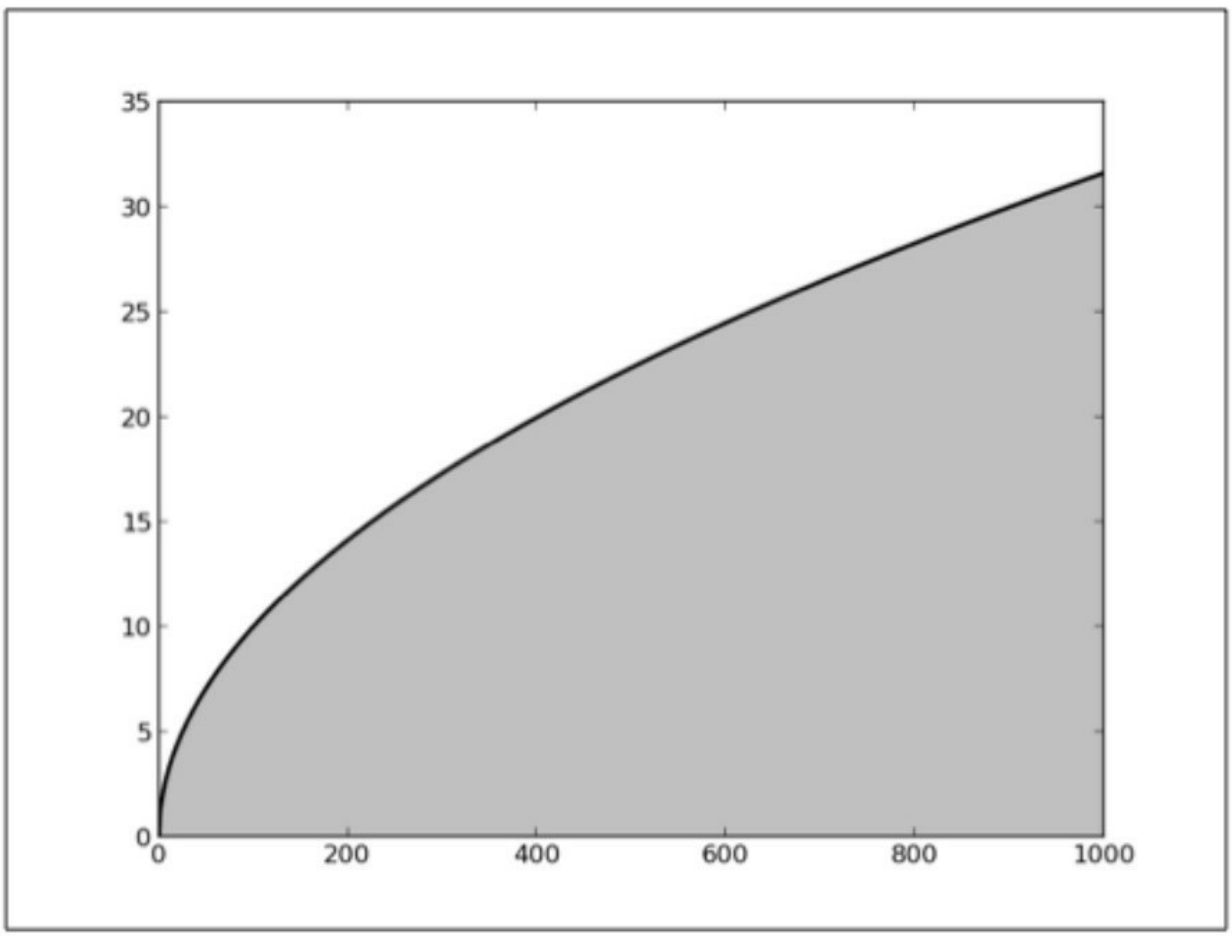


图 4- 8

在这里，我们将演示另一个技巧。它将为fill函数引入更多的条件，

示例代码如下。

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0.0, 2, 0.01)
y1 = np.sin(np.pi*x)
y2 = 1.7*np.sin(4*np.pi*x)
fig = plt.figure()
axes1 = fig.add_subplot(211)
axes1.plot(x, y1, x, y2, color='grey')
axes1.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue',
interpolate=True)
axes1.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold',
interpolate=True)
axes1.set_title('Blue where y2 <= y1. Gold-color where y2 >= y1.')
axes1.set_ylim(-2,2)
# Mask values in y2 with value greater than 1.0
y2 = np.ma.masked_greater(y2, 1.0)
axes2 = fig.add_subplot(212, sharex=axes1)
axes2.plot(x, y1, x, y2, color='black')
axes2.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue',
interpolate=True)
axes2.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold',
interpolate=True)
axes2.set_title('Same as above, but mask')
axes2.set_ylim(-2,2)
axes2.grid('on')
plt.show()
```

以上代码将渲染出如图4-9所示的图形。

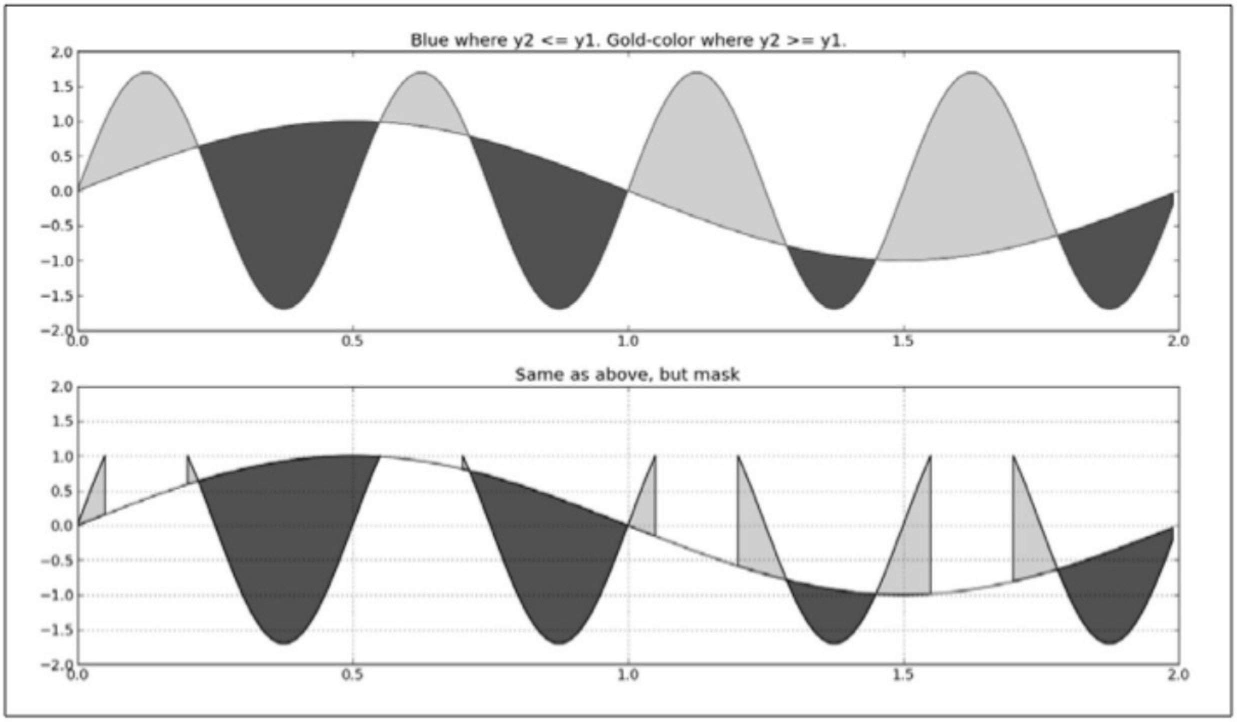


图4-9

4.8.3 工作原理

在这个例子中，首先创建了两个在某些点重叠的正弦曲线函数。

还创建了两个子区，用来比较两种渲染填充区域方式的差异。

在这两种情况下，我们使用了带参数where的fill_between()方法填充where等于True的区域，其中where参数接收一个长度为N的布尔数组。

下面的一个子区演示了 mask_greater，它屏蔽了数组中大于给定值的所有值。这是一个numpy.ma包中的方法，用来处理缺失或者无效的值。我们在底部的坐标轴上添加网格使其更直观。

4.9 绘制极线图

如果数据已经是以极坐标形式表示的，我们也可以用极线图来把它显示出来。即使数据不在极坐标内，也应该考虑把它转换成极坐标形式并在极线图上画出来。

要回答我们是否需要这样做，需要了解数据代表什么以及希望显示给用户什么。想象一下什么是用户想从图表中读到的和解码的，这通常会让我们得到最好的可视化效果。

极线图通常被用来显示本质上是射线的信息。例如，在太阳轨迹图中，我们看到放射投影的天空，触角的辐射图的辐射角度各异。可以从 http://www. astronwireless. com/topic-archives-antenna-radiation-patterns.asp了解更多的内容。

本节中将要学习如何改变图表中使用的坐标系统，并以极限坐标系统代替。

4.9.1 准备工作

为了在极限坐标下显示数据，必须有合适的数据值。在极坐标系统中，点被描述为半径距离（通常表示为r）和角度（通常表示为theta）。角度可以用弧度或者角度表示，但是matplotlib使用角度。

和 plot()函数十分相似的是，我们用 polar()函数绘制极线图。polar()函数接收两个相同长度的参数数组theta和r，分别用于角度数组和半径数组。函数也接收其他和plot()函数相同的格式化参数。

我们仍然需要告诉 matplotlib 坐标轴要在极限坐标系统中。这通过向 add_axes 或add_subplot提供polar=True参数来完成。

此外，为了设置图表中的其他属性，如半径网格或者角度，需要使用 matplotlib. pyplot.rgrids()来切换半径网格的显示或者设置标签。同样，使用 matplotlib. pyplot.thetagrid()来配置角度刻度和标签。

4.9.2 操作步骤

本节将演示如何绘制极线条，代码如下。

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt
figsize = 7
colormap = lambda r: cm.Set2(r / 20.)
N = 18 # number of bars
fig = plt.figure(figsize=(figsize,figsize))
ax = fig.add_axes([0.2, 0.2, 0.7, 0.7], polar=True)
theta = np.arange(0.0, 2 * np.pi, 2 * np.pi/N)
radii = 20 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
bars = ax.bar(theta, radii, width=width, bottom=0.0)
for r, bar in zip(radii, bars):
    bar.set_facecolor(colormap(r))
    bar.set_alpha(0.6)
plt.show()
```

上述代码段将生成如图4-10所示的图形。

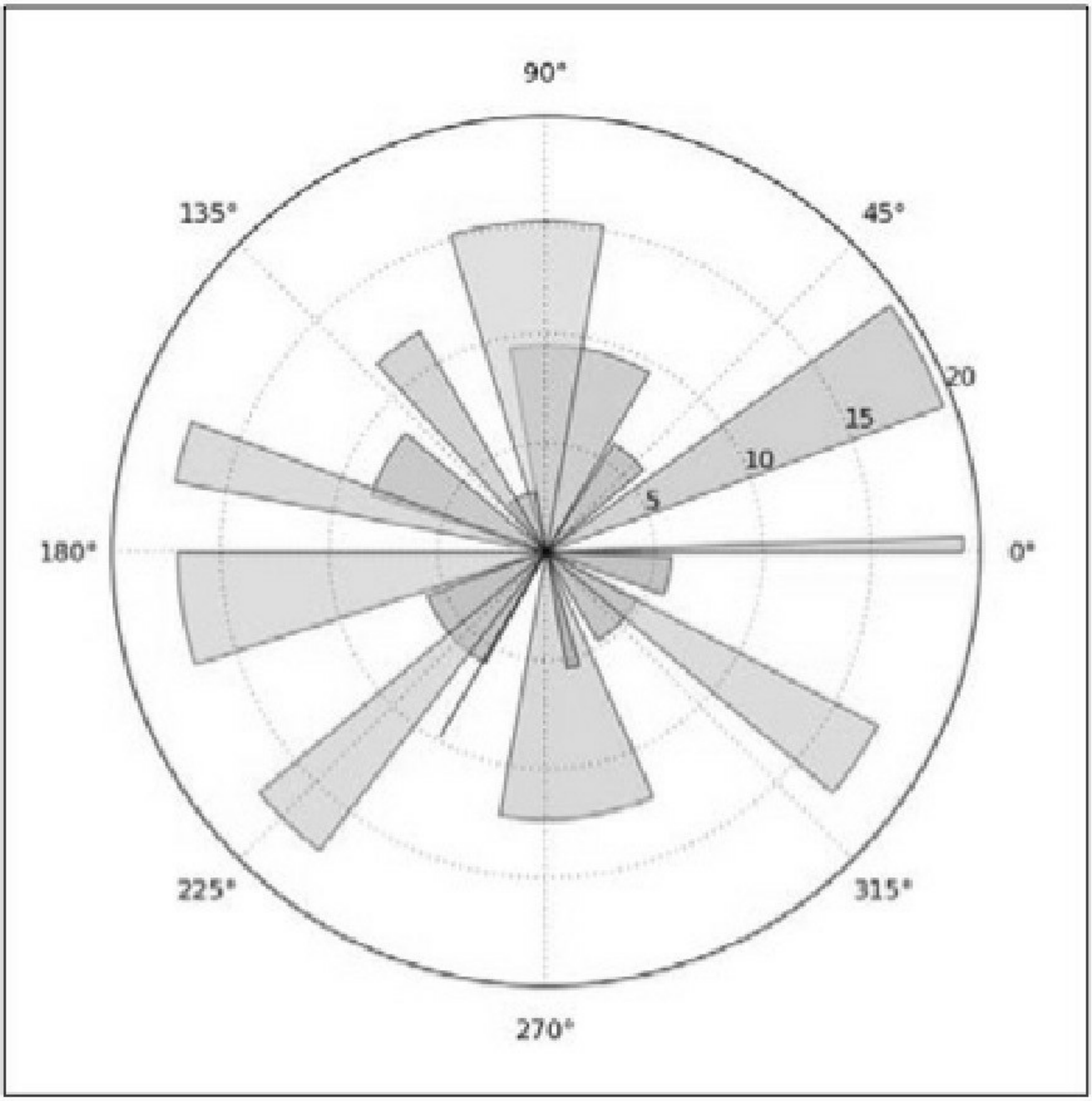


图4-10

4.9.3 工作原理

首先，创建了一个正方形的图表，并向其添加极限坐标轴。其实图表不必是正方形的，但是如果不这样的话，极线图就是椭圆形（而不是圆形）的了。

然后，为角度（theta）集合和极线距离（radii）生成随机值。因为

绘制的是极线条，需要为每一个极线条提供宽度集合，因此需要生成一些宽度值。因为matplotlib.axes.bar接收值数组（几乎matplotlib中所有的绘图函数都是如此），所以不必在这个生成的数据集合上做循环遍历，只需要调用一次bar函数，并传入所有的参数。

为了能够容易区分每一个极线条，需要循环遍历添加到ax（坐标轴）的每一个极线条，并定制化其外观（表面颜色和透明度）。

4.10 使用极线条可视化文件系统树

在本节中，我们想展示如何解决一个“现实世界”中的任务——如何用matplotlib可视化目录占有率。

本节将学习如何可视化具有比例化大小的的文件系统树。

4.10.1 准备工作

我们都有大容量的硬盘，有些时候我们都忘记里面存放的是什么了。如果能看清楚这样的大文件目录中存储的是什么，里面最大的文件是什么就好了。

虽然有许多更加复杂并且功能强大的软件产品可以完成这项工作，但是我们想用Python和matplotlib来演示一下是如何来做的。

4.10.2 操作步骤

执行下面的步骤。

1.实现一些helper函数来处理找到的文件夹和其内部的数据结构。

2.实现绘图的主函数draw()。[8]

```
import os
import sys
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
def build_folders(start_path):
    folders = []
```

```
    for each in get_directories(start_path):
        size = get_size(each)
        if size >= 25 * 1024 * 1024:
            folders.append({'size': size, 'path': each})
    for each in folders:
        print "Path: " + os.path.basename(each['path'])
        print "Size: " + str(each['size'] / 1024 / 1024) + " MB"
    return folders
def get_size(path):
    assert path is not None
    total_size = 0
    for dirpath, dirnames, filenames in os.walk(path):
        for f in filenames:
            fp = os.path.join(dirpath, f)
            try:
                size = os.path.getsize(fp)
                total_size += size
                #print "Size of '{0}' is {1}".format(fp, size)
            except OSError as err:
                print str(err)
                pass
    return total_size
def get_directories(path):
    dirs = set()
    for dirpath, dirnames, filenames in os.walk(path):
        dirs = set([os.path.join(dirpath, x) for x in dirnames])
        break # we just want the first one
```

```
    return dirs
def draw(folders):
    """ Draw folder size for given folder"""
    figsize = (8, 8) # keep the figure square
    ldo, rup = 0.1, 0.8 # leftdown and right up normalized
    fig = plt.figure(figsize=figsize)
    ax = fig.add_axes([ldo, ldo, rup, rup], polar=True)
    # transform data
    x = [os.path.basename(x['path']) for x in folders]
    y = [y['size'] / 1024 / 1024 for y in folders]
    theta = np.arange(0.0, 2 * np.pi, 2 * np.pi / len(x))
    radii = y
    bars = ax.bar(theta, radii)
    middle = 90 / len(x)
    theta_ticks = [t * (180 / np.pi) + middle for t in theta]
    lines, labels = plt.thetagrids(theta_ticks, labels=x,frac=0.5)
    for step, each in enumerate(labels):
        each.set_rotation(theta[step] * (180 / np.pi) + middle)
        each.set_fontsize(8)
    # configure bars
    colormap = lambda r:cm.Set2(r / len(x))
    for r, each in zip(radii, bars):
        each.set_facecolor(colormap(r))
        each.set_alpha(0.5)
    plt.show()
```

3.接下来，我们将实现main函数体。当从命令行调用程序时，在main函数中验证用户输入的参数。

```
if __name__ == '__main__':
    if len(sys.argv) is not 2:
        print "ERROR: Please supply path to folder."
        sys.exit(-1)
    start_path = sys.argv[1]
    if not os.path.exists(start_path):
        print "ERROR: Path must exits."
        sys.exit(-1)
    folders = build_folders(start_path)
    if len(folders) < 1:
        print "ERROR: Path does not contain any folders."
        sys.exit(-1)
    draw(folders)
```

在命令行运行下面的命令。

```
$ python ch04_rec11_filesystem.py /usr/lib/
```

生成如图4-11所示的图表。

的路径。

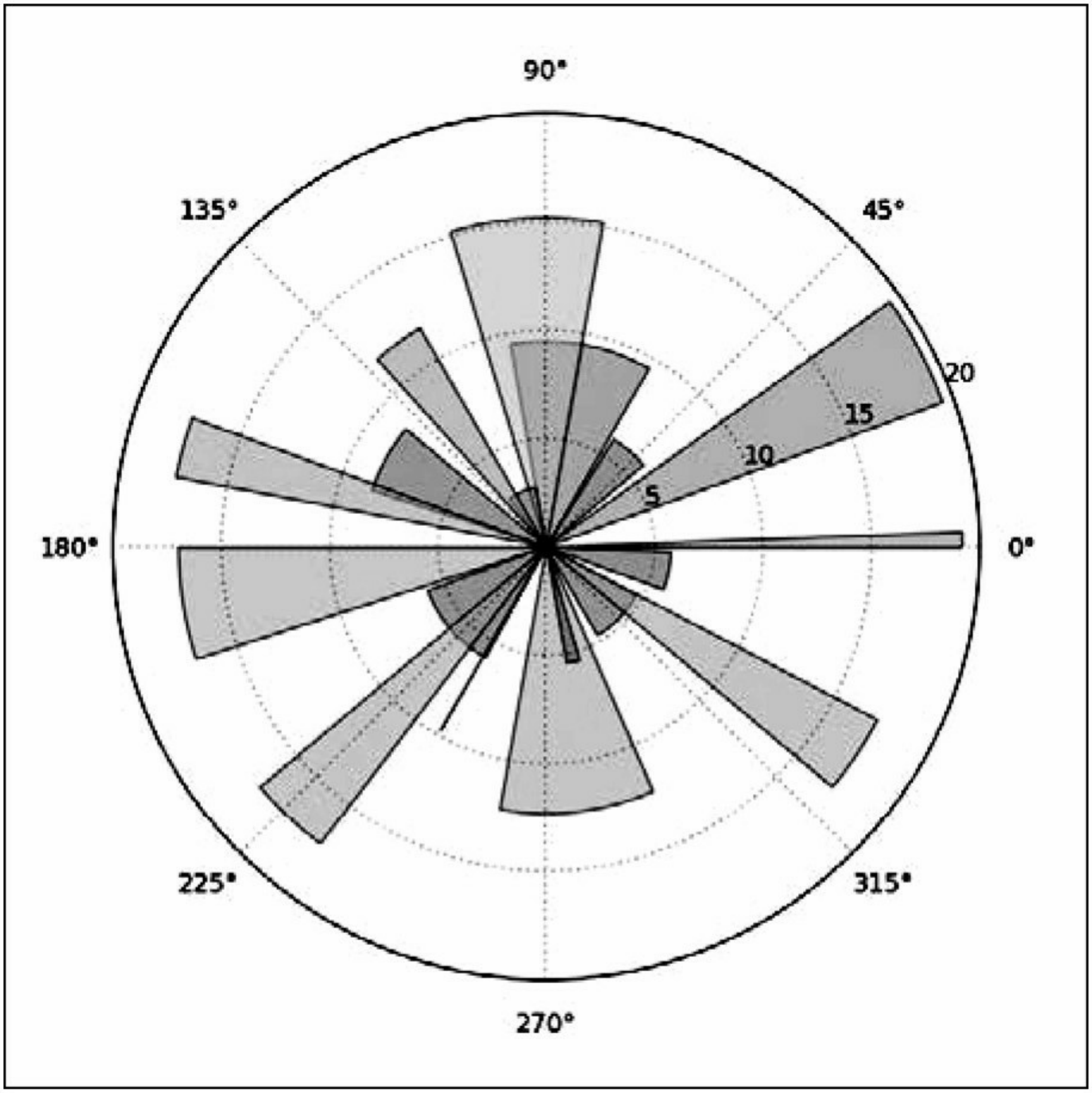


图4-11

4.10.3 工作原理

我们从代码底部 if __name__== '__main__'之后的部分开始解析，因为程序是从这里开始执行的。

使用sys模块得到命令行参数，它表示我们想要可视化的文件目录的路径。

函数 build_folders 创建出目录的列表，其中的每一项包含了在给定目录 start_path下的目录路径和大小。该方法调用get_directories返回start_path下的子目录列表。接下来，对于每一个找到的目录，用get_size函数计算出目录的字节大小。

为了调试，我们把目录打印出来以便能对图表和数据进行比较。

在创建目录列表后，把它传给函数draw。draw函数将所有数据转换到正确尺寸（这里采用极坐标系统）、创建极线图表和绘制所有极线条、刻度和标签的工作。

严格来讲，我们应该把这项工作划分成更小的函数，尤其是在对代码做进一步开发的时候。

注释

[1]. patch：直译为补丁，也可译为补片。是一个用颜色填充的图形对象。本书中采用第二种译法。

[2]. 即标题文本对象。

[3]. 原文为 1/71，应为作者笔误。

[4]. 原文为 im1，im2，im3，应为作者笔误。

[5].可以理解为 z=f(x,y)，x 和 y 为函数的两个参数，z 为函数返回值，相同的 z 值连成的曲线即为本节所讲的等高线。

[6]. 此处所指应为 process_signals 函数。

[7]. 原文为 matplotlib.pyploy.fill_betweenx()，应为作者笔误。

[8]. 原文在步骤 2 后面还有一步，但和译文中的步骤 3 重复，因此译者将其去掉了。

第5章 创建3D可视化图表

本章将学习以下内容。

◆ 创建 3D 柱状图

◆ 创建 3D 直方图

◆ 在 matplotlib 中创建动画

◆ 用 OpenGL 制作动画

5.1 简介

3D可视化有时候是很有效的，有时候也是不可避免的。在这里我们将展示一些例子，这些例子将满足一些最常用的需求。

本章将会介绍并讲解一些3D可视化的话题。

5.2 创建 3D 柱状图

虽然matplotlib主要专注于绘图，并且主要是二维的图形，但是它也有一些不同的扩展，能让我们在地理图上绘图，让我们把Excel和3D图表结合起来。在matplotlib的世界里，这些扩展叫做工具包（toolkits）。工具包是一些关注在某个话题（如3D绘图）的特定函数的集合。

比较流行的工具包有 Basemap、GTK 工具、Excel 工具、Natgrid、AxesGrid 和 mplot3d。

本节将探索关于 mplot3d 的更多功能。mpl_toolkits.mplot3工具包提供了一些基本的3D绘图功能，其支持的图表类型包括散点图（scatter）、曲面图（surf）、线图（line）和网格图（mesh）。虽然mplot3d不是一个最好的3D图形绘制库，但是它是伴随着matplotlib产生的，因此我们对其接口已经很熟悉了。

5.2.1 准备工作

基本来讲，我们仍然需要创建一个图表并把想要的坐标轴添加到上面。但不同的是我们为图表指定的是3D视图，并且添加的坐标轴是Axes3D。

现在，我们可以使用几乎相同的函数来绘图了。当然，函数的参数是不同的，需要为3个坐标轴提供数据。

例如，我们要为函数 mpl_toolkits.mplot3d.Axes3D.plot 指定 xs、ys、zs 和zdir 参数。其他的参数则直接传给 matplotlib.axes.Axes.plot。下面来解释一下这些特定的参数。

1.xs和ys：x轴和y轴坐标。

2.zs：这是z轴的坐标值，可以是所有点对应一个值，或者是每个点对应一个值。

3.zdir：决定哪个坐标轴作为z轴的维度（通常是zs，但是也可以是xs或者ys）。



模块 mpl_toolkits.mplot3d.art3d包含了 3D artist 代码和将 2Dartists转化为3D版本的函数。在该模块中有一个rotate_axes方法，该方法可以被添加到Axes3D中来对坐标重新排序，这样坐标轴就与zdir一起旋转了。zdir默认值为z。在坐标轴前加一个'-'会进行反转转换，这样一来，zdir的值就可以是x、-x、y、-y、z或者-z。

5.2.2 操作步骤

以下代码演示了我们所解释的概念。

```
import random
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from mpl_toolkits.mplot3d import Axes3D
mpl.rcParams['font.size'] = 10
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for z in [2011, 2012, 2013, 2014]:
    xs = xrange(1,13)
    ys = 1000 * np.random.rand(12)
```

```
    color = plt.cm.Set2(random.choice(xrange(plt.cm.Set2.N)))
    ax.bar(xs, ys, zs=z, zdir='y', color=color, alpha=0.8)
ax.xaxis.set_major_locator(mpl.ticker.FixedLocator(xs))
ax.yaxis.set_major_locator(mpl.ticker.FixedLocator(ys))
ax.set_xlabel('Month')
ax.set_ylabel('Year')
ax.set_zlabel('Sales Net [usd]')
plt.show()
```

上述代码生成如图5-1所示的图表。

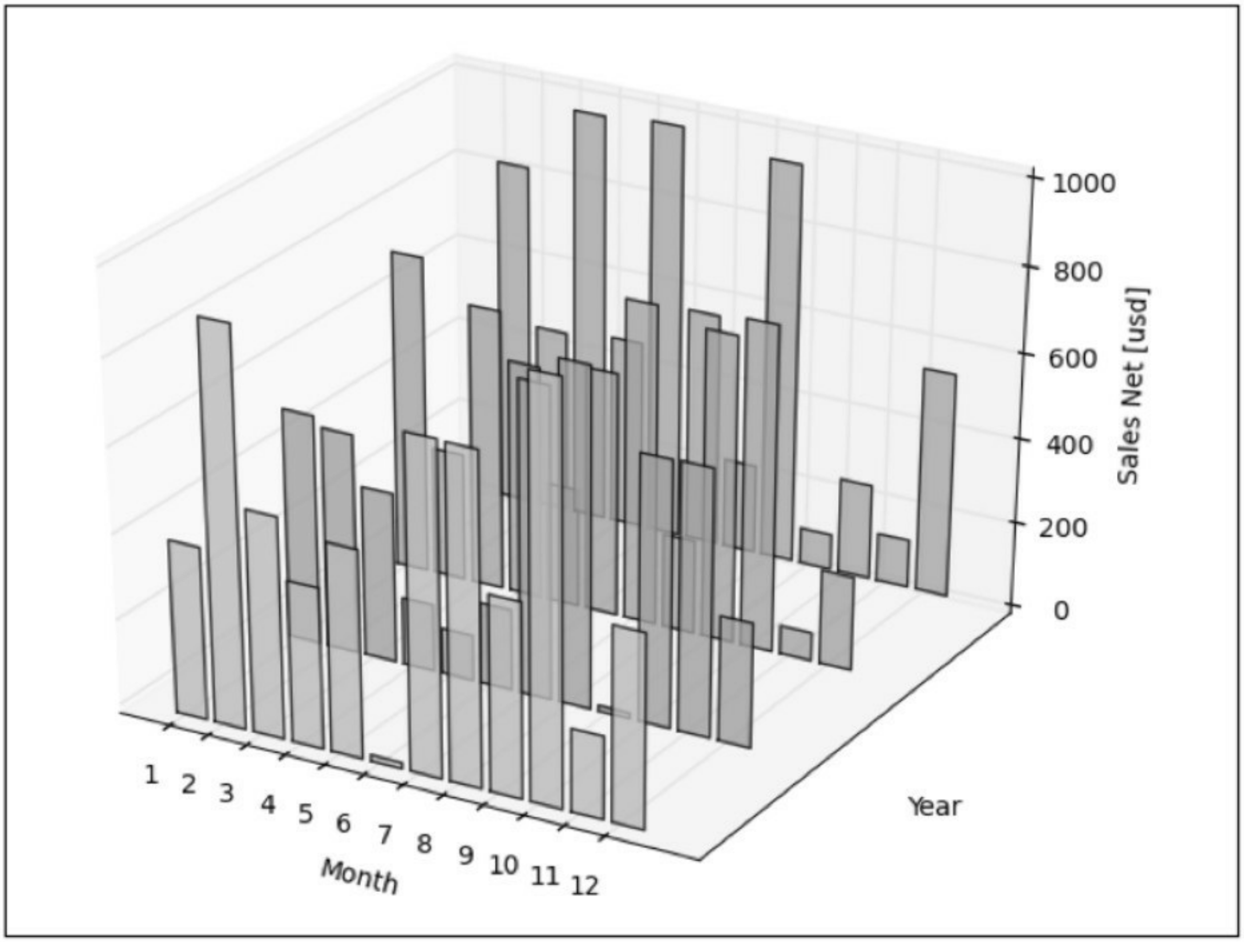


图5-1

5.2.3 工作原理

我们需要像在 2D 世界中那样做相同的准备工作。不同的是，在这里需要指定后端（backend）的种类。然后生成了一些随机数据，例如4年的销售额（2011-2014）。

我们需要为3D坐标轴指定相同的Z值。

从颜色映射集合中随机选择一种颜色，然后把它和每一个Z-order集合的xs、ys对关联起来。最后，用xs、ys对渲染出柱状条序列。

5.2.4 补充说明

其他的一些matplotlib的2D绘图函数在这里也是可以用的，例如scatter()和plot()有着相似的接口，但有额外的点标记大小参数。我们对contour、contourf和bar也非常熟悉。

仅在 3D 中出现的新图表类型有线框图（wireframe）、曲面图（surface）和三翼面图（tri-surface）。

在下面的示例代码中，我们绘制了著名的Pringle函数的三翼面图，数学专业上的叫法是双曲面抛物线（hyperbolic paraboloid）。

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
n_angles = 36
n_radii = 8
# An array of radii
# Does not include radius r=0, this is to eliminate duplicate points
radii = np.linspace(0.125, 1.0, n_radii)
# An array of angles
angles = np.linspace(0, 2 * np.pi, n_angles, endpoint=False)
```

```
# Repeat all angles for each radius
angles = np.repeat(angles[..., np.newaxis], n_radii, axis=1)
# Convert polar (radii, angles) coords to cartesian (x, y) coords
# (0, 0) is added here. There are no duplicate points in the (x, y)
plane
x = np.append(0, (radii * np.cos(angles)).flatten())
y = np.append(0, (radii * np.sin(angles)).flatten())
# Pringle surface
z = np.sin(-x * y)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
plt.show()
```

上面的代码生成如图5-2所示的图形。

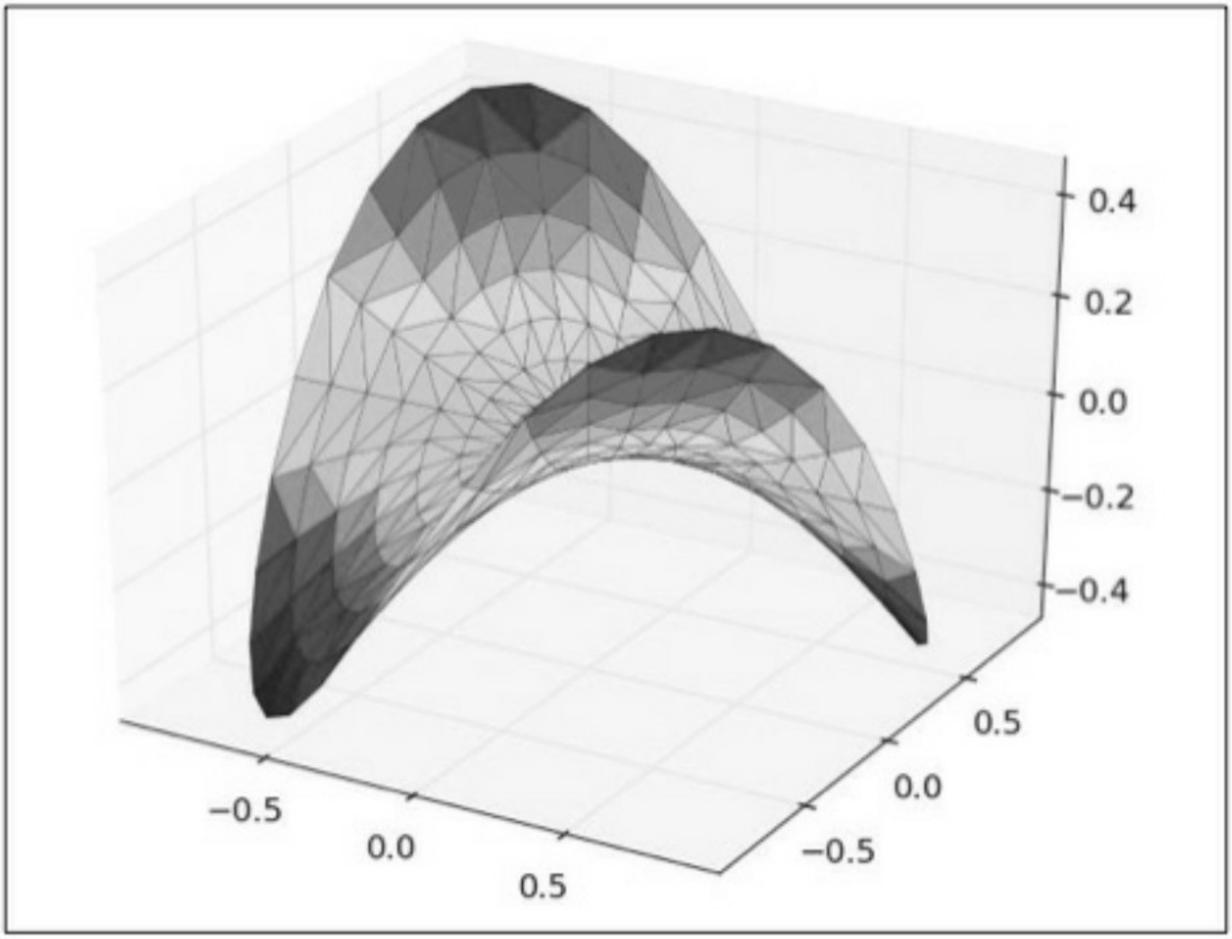


图5-2

5.3 创建 3D 直方图

像3D柱状图一样，我们可能想创建3D直方图。3D直方图可以用来很容易地识别3个独立变量之间的相关性。可以用它们来从图像中提取信息，其中第三个维度可以是所分析的图像的（x, y）空间通道的强度。

本节将学习如何创建3D直方图。

5.3.1 准备工作

回顾一下，直方图表示的是一些值在特定列（通常叫做“bin”）中的发生率。那么，三维直方图表示的是在一个网格中的发生率。网格是矩形的，表示的是在两列中关于两个变量的发生率。

5.3.2 操作步骤

在这个计算过程中，我们将进行如下操作。

1.使用Numpy，因为其拥有计算两个变量的直方图的函数。

2.用正态分布函数生成x和y，但是给它们提供不同的参数，以便能区分结果直方图的相互关系。

3.用相同的数据集合绘制散点图，展示散点图和3D直方图显示上的差异。

下面是实现上述步骤的代码。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
```

```
from mpl_toolkits.mplot3d import Axes3D
mpl.rcParams['font.size'] = 10
samples = 25
x = np.random.normal(5, 1, samples)
y = np.random.normal(3, .5, samples)
fig = plt.figure()
ax = fig.add_subplot(211, projection='3d')
# compute two-dimensional histogram
hist, xedges, yedges = np.histogram2d(x, y, bins=10)
# compute location of the x,y bar positions
elements = (len(xedges) - 1) * (len(yedges) - 1)
xpos, ypos = np.meshgrid(xedges[:-1]+.25, yedges[:-1]+.25)
xpos = xpos.flatten()
ypos = ypos.flatten()
zpos = np.zeros(elements)
# make every bar the same width in base
dx = .1 * np.ones_like(zpos)
dy = dx.copy()
# this defines the height of the bar
dz = hist.flatten()
ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', alpha=0.4)
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')
# plot the same x,y correlation in scatter plot
# for comparison
ax2 = fig.add_subplot(212)
```

```
ax2.scatter(x, y)
ax2.set_xlabel('X Axis')
ax2.set_ylabel('Y Axis')
plt.show()
```

上述代码生成如图5-3所示的图形。

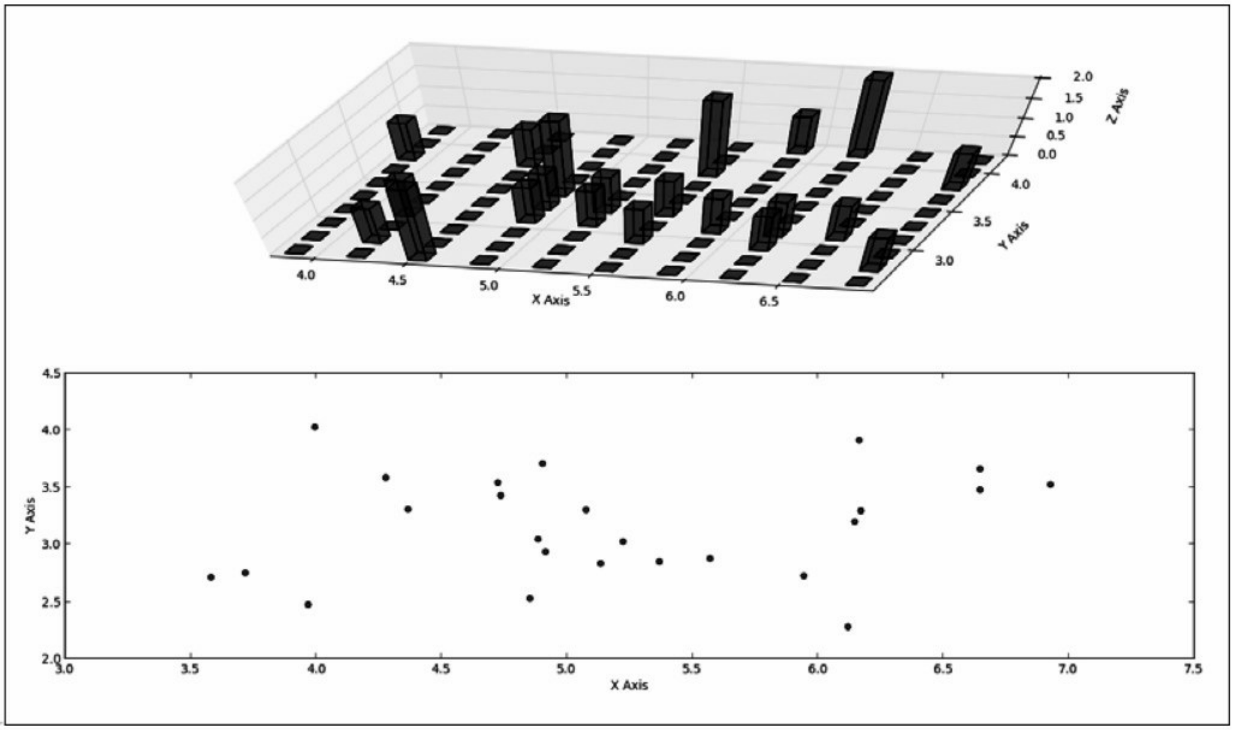


图5-3

5.3.3 工作原理

我们用 np.histogram2d 生成了一个直方图，该方法返回了直方图（hist）、x bin边界和 y bin边界。

bar3d 函数需要 x, y 空间的坐标，因此需要计算出一般的矩阵坐标，对此我们使用np.meshgrid函数把x和y位置的向量合并到2D空间网格中（矩阵）。我们可以使用它在xy平面位置上绘制矩形条。

变量dx和dy表示每一个矩形条底部的宽度，我们想把它设置为常

数，因此我们为xy平面的每一个位置给定的值为 0.1 个点的宽度。

z轴上的值（dz）实际上是计算机直方图（在变量hist中），它表示在一个特定的bin中一般的x和y样本的个数。

接下来在散点图（图5-3）中显示了一个2D坐标轴，也呈现了两组相似但起始参数不同的分布间的相互关系。

有时候，3D给予我们更多的信息，并以一个更好的方式让我们来理解数据所包含的内容。然而在更多情况下，3D可视化比2D更加让人感到迷惑，所以在舍弃2D选择3D之前最好慎重考虑。

5.4 在matplotlib中创建动画

本节将学习如何让图表动起来。有时候，在解释当我们改变变量值时会发生什么情况的时候，动画有着更强的描述性。主要函数库的动画能力有限，但通常已足够了。接下来将解释如何使用它们。

5.4.1 准备工作

从 1.1 版本开始，一个动画框架被添加到了标准 matplotlib 库中，该框架主要的类是matplotlib.animation.Animation。这个类是一个基类，它可以针对不同的行为被子类化。实际上，该框架已经提供了几个类：TimedAnimation、ArtistAnimation 和FuncAnimation。表5-1给出了这几个类的描述。

表5-1

类名（父类）	描　　述
Animation(object)	此类用 matplotlib 创建动画。它仅仅是一个基类，应该被子类化以提供所需的行为
TimedAnimation(Animation)	这个动画子类支持基于时间的动画，每 interval*milliseconds 绘制一个新的帧
ArtistAnimation(TimedAnimation)	在调用此函数之前，所有绘制工作应当已经完成，并且相关的 artists 已经被保存
FuncAnimation(TimedAnimation)	其通过重复地调用一个函数生成动画，可以为函数传入参数，参数是可选的

为了能把动画存储到一个视频文件中，必须安装ffmpeg或者mencoder。这些包的安装根据我们所使用的操作系统的不同以及不同版

本间的差别会有所不同，因此我们把它留给亲爱的读者去Google一下有效的相关信息。

5.4.2 操作步骤

下述代码演示了一些matplotlib动画。

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation
fig = plt.figure()
ax = plt.axes(xlim=(0, 2), ylim=(-2, 2))
line, = ax.plot([], [], lw=2)
def init():
    """Clears current frame."""
    line.set_data([], [])
    return line,
def animate(i):
    """Draw figure.
    @param i: Frame counter
    @type i: int
    """
    x = np.linspace(0, 2, 1000)
    y = np.sin(2 * np.pi * (x - 0.01 * i)) * np.cos(22 * np.pi * (x - 0.01 *
i))
    line.set_data(x, y)
    return line,
# This call puts the work in motion
```

```
# connecting init and animate functions and figure we want to draw
animator = animation.FuncAnimation(fig, animate, init_func=init,
        frames=200, interval=20, blit=True)
# This call creates the video file.
# Temporary, every frame is saved as PNG file
# and later processed by ffmpeg encoder into MPEG4 file
# we can pass various arguments to ffmpeg via extra_args
animator.save('basic_animation.mp4', fps=30,
    extra_args=['-vcodec', 'libx264'],
    writer='ffmpeg_file')
plt.show()
```

本代码将在执行该文件的文件夹中创建文件 basic_animation.mp4，同时显示一个有动画的图形窗口。该视频文件可以用大多数支持MPEG-4格式的视频播放器打开。图形（帧）看上去如图5-4所示。

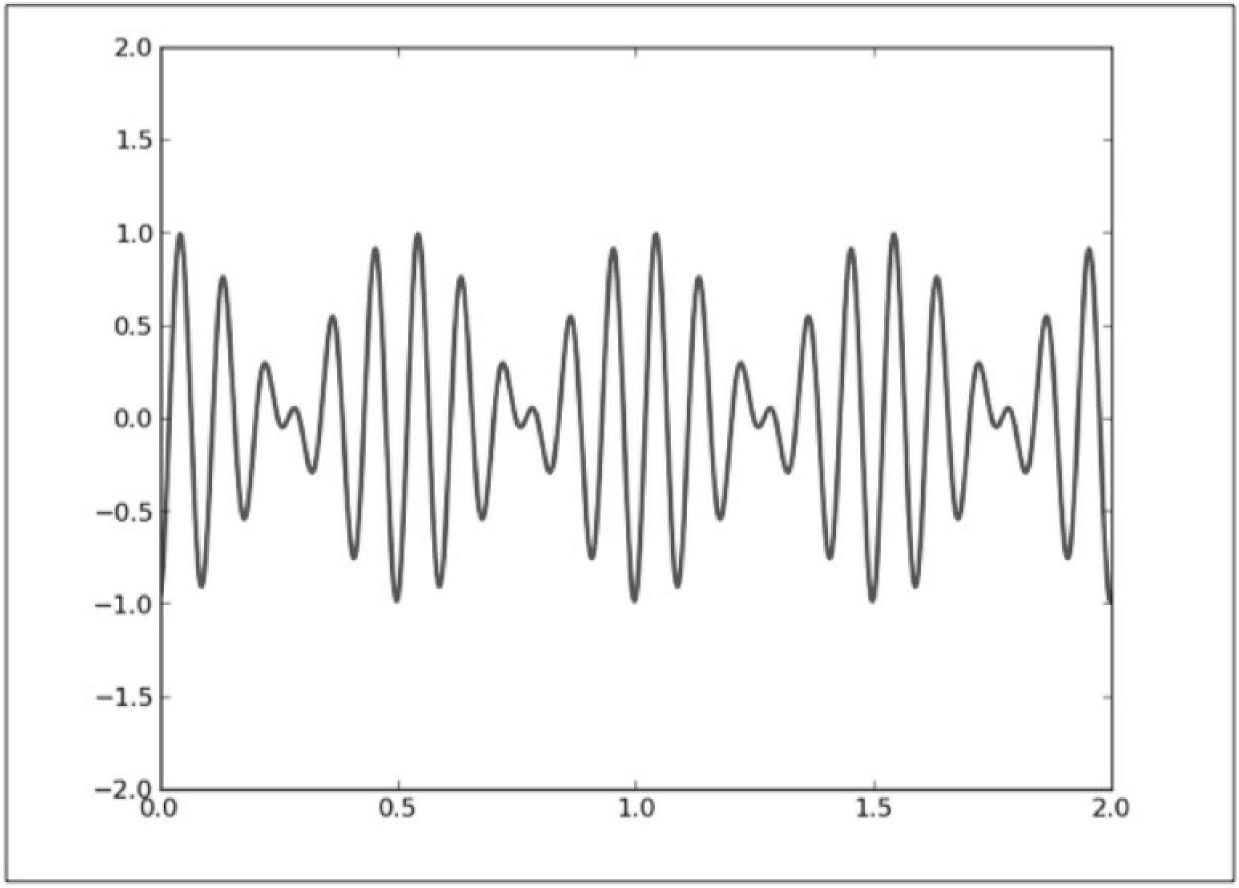


图 5- 4

5.4.3 工作原理

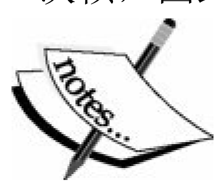
上面例子中最重要的几个函数是 init()、animate()和 save()。首先，通过向FuncAnimate[1]传入两个回调函数，init和animator。然后，调用它的save（）方法保存视频文件。表5-2是关于每一个函数更多的细节内容。

表5-2

函　数　名	用　　　法
init	通过参数 init_func 传入 matplotlib.animation.FuncAnimation 构造器中，在绘制下一帧前清空当前帧
animate	通过参数 func 传入 matplotlib.animation.FuncAnimation 构造器中。通过 fig 参数传入想要绘制动画的图形窗口，其内部实际上是将 fig 传入到 matplotlib.animation.FuncAnimation 构造器中，把要绘制图形的窗口和动画事件关联起来。该函数从 frames，通常是表示许多帧的迭代器获取（可选的）参数
matplotlib.animation.Animation.save	通过绘制每一帧保存一个视频文件。在通过编码器（ffmpeg 或者 mencoder）创建一个视频文件之前，先创建临时图像文件。该方法也接收各种参数来配置视频输出、元数据（如作者等）、使用的编码器、分辨率/大小，等等。其中一个参数是用来指定使用何种视频编码器，目前支持的类型有 ffmpeg、ffmpeg_file 和 mencoder

5.4.4 补充说明

matplotlib.animation.ArtistAnimation的用法和FuncAnimation不同，我们必须事先绘制出每一个artist，然后用所有artist的不同帧来实例化ArtistAnimation类。Artist动画是对matplotlib.animation.TimedAnimation类的一种封装，每N毫秒绘制一次帧，因此它支持基于时间的动画。



不幸的是，对于 Mac OS X 的用户来说，动画框架在该平台上却让人很苦恼，有时候甚至不能工作。这在matplotlib未来的版本中会有所改进。

5.5 用OpenGL制作动画

使用OpenGL的动机来源于CPU处理能力的限制，限制体现在当我们面临一项要可视化成千上万个数据点的工作，并且要求其快速执行（有时甚至是实时的）的时候。

现代计算机拥有强大的GPU用于加速与可视化相关的计算（比如游戏）。它们没有理由不能用于科学相关的可视化。

实际上，编写硬件加速的软件至少有一个缺点。就硬件的依赖而言，现代图形卡要求有专有的驱动，有时候驱动在目标平台/机器（例如用户的笔记本）上是无法使用的；即使是可用的，有时候你也不想呆在那花大把的时间去安装驱动所依赖的软件，相反，你想把时间花费在展示你的发现，并演示你的研究成果上。虽然这并不会成为编写硬件加速软件的障碍，但是你还是需要考虑一下这件事情，并且衡量一下在项目中引入这个复杂性的成本和收益。

解释完缺点后，我们可以对硬件加速可视化说“是”，可以对OpenGL，这一图形加速的工业标准说“是”。

我们将使用OpenGL来完成本节的内容，因为它是跨平台的，因此所有的例子应该在Linux、Mac 或者 Windows 上都是工作的，就像我们所演示的那样。这里假定你已经安装了所需的硬件和操作系统级别的驱动。

5.5.1 准备工作

如果你从来没有使用过 OpenGL，现在我们将做一个快速的介绍来帮助你理解。但是要真正的了解OpenGL，至少要阅读并理解一整本

书。OpenGL是一个规范，而不是一个实现，因此OpenGL本身并没有任何实现代码，所有的实现是遵循该规范而开发的库。这些库是跟随你的操作系统，或者由如NVIDIA或者AMD/ATI等不同的显卡厂商发布的。

此外，OpenGL只关注图形渲染而不是动画、定时和其他复杂的事情，这些事情是留给其他库来完成的。



OpenGL动画基础

因为OpenGL是一个图形渲染库，所以它不知道我们在屏幕上绘制的是什么。它不关心我们画的是否是一只猫、一个球，或者一条线，还是所有这些对象。因此，要移动一个已经渲染的对象，需要清除并重绘整个图像。为了让某个物体动起来，我们需要很快地循环绘制和重绘所有内容，并把它显示给用户，这样用户就认为他/她正在观看一个动画。

在机器上安装 OpenGL 是一件和平台相关的过程。在 Mac OS X 上，OpenGL 的安装通过系统升级来完成，但是开发库（所谓的“头文件”）是Xcode开发包的一部分。

在Windows系统上，最好的方式是安装电脑的显卡厂商的最新显卡驱动程序。OpenGL可能并不需要它们就可以工作，但那样的话你就很可能失去了原版驱动程序的最新特性。

在 Linux 平台上，如果你不反对安装闭源软件，在操作系统发行版自身的软件管理器中，或者显卡厂商网站上的二进制安装文件，都提供了可供下载的特定厂商的驱动。Mesa3D几乎一直都是OpenGL的标准实现，它也是最有名的OpenGL实现，使用Xorg来为Linux、FreeBSD和类似操作系统的OpenGL提供支持。

基本上，在Debian/Ubuntu系统中，应当安装下列软件包及其依赖。

```
$ sudo apt-get install libgl1-mesa-dev libgl-mesa-dri
```

然后，你就可以使用一些开发库和/或者框架来实际地编写OpenGL支持的应用程序了。

我们在这里只关注Python中的OpenGL绘图，因此我们将回顾在Python中使用最多的一些构建在OpenGL之上的库和框架。我们会提到matplotlib及其当前和将来对OpenGL的支持。

◆ Mayavi：这是一个专门用于 3D 的库。

◆ Pyglet：这是一个纯 Python 的图形库。

◆ Glumpy：这是一个构建在 Numpy 之上的快速图形渲染库。

◆ Pyglet 和 OpenGL：这是用来可视化大数据（百万级数据点）的。

5.5.2 操作步骤

专业化的项目Mayavi是一个功能全面的3D图形库，它主要用于高级3D渲染。它包含在已经提到的 Python 包中，如 EPD（虽然没有免费许可）。这也是在 Windows 和 Mac OS X操作系统上的推荐安装方式。在Linux平台上，也可以通过pip轻松地安装，代码如下。

```
$ pip install mayavi
```

Mayavi 可以作为一个开发库/框架，或者一个应用程序来使用。Mayavi 应用程序包含了一个可视化编辑器，可以用于简单的数据研究和一些交互可视化。

作为一个图形库，Mayavi的用法和matplotlib相似。它可以从一个脚本接口中，或者作为一个完全的面向对象的库来使用。Mayavi的大多数接口在mlab模块中，可以使用它们来制作动画。例如，可以像下面代码那样来完成一个简单的Mayavi动画。

```
import numpy
```

```
from mayavi.mlab import *
# Produce some nice data.
n_mer, n_long = 6, 11
pi = numpy.pi
dphi = pi/1000.0
phi = numpy.arange(0.0, 2*pi + 0.5*dphi, dphi, 'd')
mu = phi*n_mer
x = numpy.cos(mu)*(1+numpy.cos(n_long*mu/n_mer)*0.5)
y = numpy.sin(mu)*(1+numpy.cos(n_long*mu/n_mer)*0.5)
z = numpy.sin(n_long*mu/n_mer)*0.5
# View it.
l = plot3d(x, y, z, numpy.sin(mu), tube_radius=0.025,
colormap='Spectral')
# Now animate the data.
ms = l.mlab_source
for i in range(100):
    x = numpy.cos(mu)*(1+numpy.cos(n_long*mu/n_mer +
        numpy.pi*(i+1)/5.)*0.5)
    scalars = numpy.sin(mu + numpy.pi*(i+1)/5)
    ms.set(x=x, scalars=scalars)
```

上述代码将生成如图5-5所示的带旋转图形的窗口。

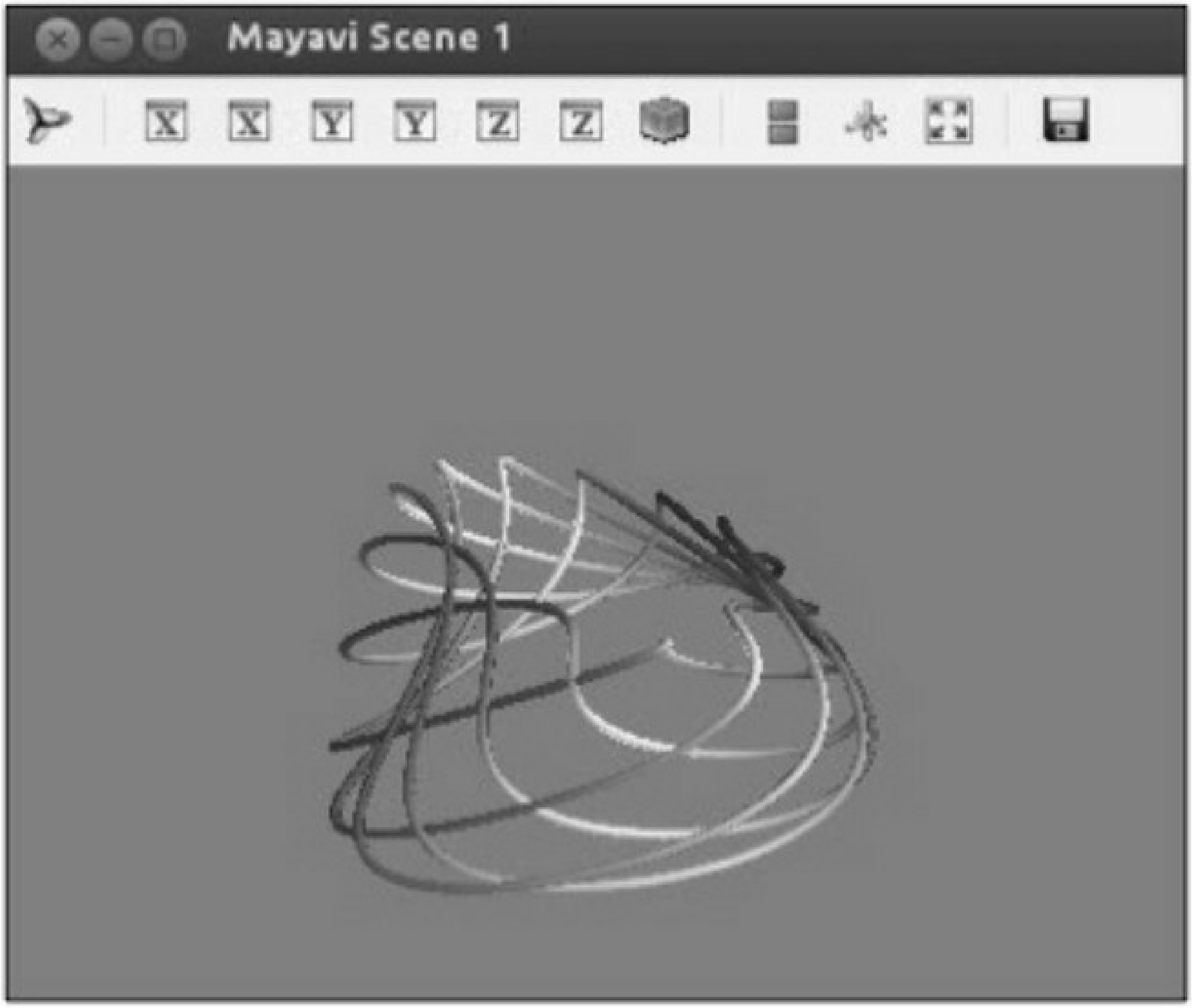


图5-5

5.5.3 工作原理

我们生成了数据集合，并创建了x、y和z三个函数。这些函数被用在plot3d函数中作为图形的起始位置。

然后，导入 mlab_source 对象，以便能在点和标量的级别上操作图形。然后使用这个特性在循环中设置特定的点和标量来创建一个100帧的旋转动画。

5.5.4 补充说明

如果你想实验更多的内容，最简单的方式是打开IPython，导入myayvi.lab，并运行一些名字为test_*的函数。

为了了解到底发生了什么，你可以借助IPython的功能来检查和研究Python源码，像下面代码显示的这样。

```
In [1]: import mayavi.mlab
In [2]: mayavi.mlab.test_simple_surf??
Type:       function
String Form:<function test_simple_surf at 0x641b410>
File:       /usr/lib/python2.7/dist-packages/mayavi/tools/helper_
functions.py
Definition: mayavi.mlab.test_simple_surf()
Source:
def test_simple_surf():
    """Test Surf with a simple collection of points."""
    x, y = numpy.mgrid[0:3:1,0:3:1]
    return surf(x, y, numpy.asarray(x, 'd'))
```

这里，我们看到如何通过在函数名后面添加两个问号（“??”）让IPython找到函数的源码并显示。这是一个真实的探索性计算，经常在可视化社区中被使用，因为它是了解数据和代码的一个快速的方式。

Pyglet快速入门

Pyglet是另一个著名的Python库，可以让编写图形和与窗口相关的应用程序变得轻松起来。它通过模块pyglet.gl来支持OpenGL，但是为了能使用Pyglet的威力你不必直接使用这个模块。通过pyglet.graphics来使用它是最方便的用法。

Pyglet采用了一种和Mayavi不同的方式。它没有可视化的IDE，你要负责从创建窗口，到发出一个低级别的 OpenGL 调用来配置 OpenGL上下文环境的所有工作。它有时比Mayavi慢，但是你所获得的是控制应

用程序的每个部分的能力。这有时候也意味着会投入更多的工作时间，但是通常来讲，它也意味着你的应用程序有更高的质量和性能。

可以通过下面的代码来得到一个简单的应用程序（图像查看器）。

```
import pyglet
window = pyglet.window.Window()
image = pyglet.resource.image('kitten.jpg')
@window.event
def on_draw():
    window.clear()
    image.blit(0, 0)
pyglet.app.run()
```

上述代码创建了一个窗口，加载了一幅图像，并指定了当我们绘制一个窗口对象时所发生的事件（换言之，我们为 on_draw事件定义了一个事件处理器）。最后，运行我们的程序（piglet.app.run()）。

在实现的内部，程序使用OpenGL在窗口上进行绘制。此接口可以从pyglet.gl模块获得。然而直接使用它是不高效的，因此pyglet在pyglet.graphics中提供了一个更简单的接口，在这个接口内部使用了顶点数组（vertex arrays）和缓冲区（buffers）。

Glumpy快速入门

Glumpy是一个OpenGL+NumPy库，它用OpenGL来进行快速Numpy可视化。它是一个由 Nicolas Rougier 启动的开源项目，致力于高效可视化。为了使用它，我们需要 Python OpenGL绑定（bindings）、SciPy，当然还有Glumpy。安装命令如下。

```
sudo apt-get install python-opengl
sudo pip install scipy
sudo pip install glumpy
```

Glumpy 使用 OpenGL 纹理（textures）来表示阵列，因为这恐怕是

在现代图形硬件上最快的可视化方法了。

Pyprocessing 简介

Pyprocessing和Processing（http://processing.org）的工作方式极其相似。Pyprocessing中的大多数函数和Processing函数是相同的。如果你熟悉Processing和Python，你就已经知道了编写Pyprocessing程序所需的几乎所有知识。为了使用它，我们唯一需要做的事情是导入pyprocessing包，用Pyprocessing函数和数据结构来编写剩余的代码，然后调用run()函数来执行。

有很多关于 OpenGL 以及如何在 C/C++或者任何其他语言 binding 中使用它的免费教程。在 OpenGL 官方 wiki 上提供了一个清单，地址为 http://www.opengl.org/wiki/Getting_started#Tutorials_and_How_To_Guides。

总之，还有许多处理Python、OpenGL和3D可视化的项目，其中有一些比较年轻，有一些已经不再维护了，但是如果你发现有项目应该被提到，请告诉我们。

注释

[1]. 应为 FuncAnimation。

第6章 用图像和地图绘制图表

本章将学习以下内容。

◆ 用 PIL 做图像处理

◆ 绘制带图像的图表

◆ 在带其他图形的图表中显示图像

◆ 使用 Basemap 在地图上绘制数据

◆ 使用 Google Map API 在地图上绘制数据

◆ 生成 CAPTCHA 图像

6.1 简介

本章将探索如何使用图像和地图来一起协同工作。Python有一些著名的图像库，允许我们以美学和科学的方式处理图像。

我们将演示如何通过应用滤波器和调整图像大小来进行图像处理，以此来了解PIL的能力。

另外，我们将展示如何把图像文件作为matplotlib图表的注解（annotation）。

为了处理地理数据集合的数据可视化，我们将学习Python的可用库和公开API的功能，并将其应用于基于地图的视觉呈现中。

在最后一节中我们将展示用Python如何创建CAPTCHA测试图像。

6.2 用PIL做图像处理

如果我们能用WIMP（http://en.wikipedia.org/wiki/WIMP_(computing)）或者WYSIWYG （http://en.wikipedia.org/wiki/WYSIWYG）来达到相同的目的，为什么要使用Python来做图像处理呢？原因是我们想要创建一个自动化的系统来实时地处理图像，而不需要人的参与，进而优化图像处理的流程。

6.2.1 准备工作

请注意，PIL坐标系统假定坐标（0，0）位于左上角。

Image模块有一个非常有用的类和一些实例方法来对加载的图像对象（im）执行基本的操作。

◆ im = Image.open(filename):打开一个文件，并把图像加载到 im对象上。

◆ im.crop(box): 裁剪 box.box定义的左、上、右、下像素坐标（例如 box = (0, 100, 100, 100)）指定的坐标区域内的图像。

◆ im.filter(filter):为图像应用一个滤波器，并返回滤波后的图像。

◆ im.histogram():返回该图像的直方图列表，其中的每一个元素代表像素值。对于单通道图像，列表中的元素数目为 256，但是如果图像不是单通道图像，列表中会包含更多元素。对于RGB图像，列表包含768个元素（每个通道有256个值）。

◆ im.resize(size, filter):重新调整图像大小，并且使用一个滤波器进行重新采样（resampling）。可能的滤波器有NEAREST、BILINEAR、

BICUBIC和ANTIALIAS。默认值为NEAREST。

◆ im.rotate(angle, filter):逆时针方向旋转图像。

◆ im.split():分离图像波段（band）并返回一个单一波段的元组。这对于分离一个RGB图像为3个单独的波段图像非常有用。

◆ im.transform(size, method, data, filter):用 data 和 filter 对一个给定的图像做转换，转换类型可以是AFFINE、EXTENT、QUAD和MESH。可以在官方文档中了解更多关于转换的内容。Data设定了原始图像中转换被应用的区域。

ImageDraw模块允许我们在图像上绘图，可以用arc、ellipse、pieslice、point和polygon等函数来修改所加载图像的内容。

ImageChops模块包含一些图像通道操作函数（因此命名为Chops），这些函数可以被用于图像合成、着色、特效以及其他处理操作。通道操作仅限于8比特的图像。下面是一些有趣的通道操作。

◆ ImageChops.duplicate(image)：拷贝当前图像到一个新的图像对象。

◆ ImageChops.invert(image)：反转一幅图像并返回一个副本。

◆ ImageChops.difference(image1, image2)：在不用目测的情况下验证两幅图是否相同时非常有用。

ImageFilter 模块包含了卷积核（convolution kernels）类的实现，这些类允许我们创建定制化的卷积核。模块还包含了一些功能健全的常用滤波器，我们能在图像上应用这些著名的滤波器（BLUR和MedianFilter）。

ImageFilter 模块提供了两种过滤器：固定的图像增强滤波器和需要指定参数的图像滤波器，例如，把要使用的核大小作为参数。



在IPython中可以很容易地得到所有固定的滤波器的名字，代码如下。

```
In [1]: import ImageFilter
In [2]: [f for f in dir(ImageFilter) if f.isupper()]
Out[2]:
['BLUR',
'CONTOUR',
'DETAIL',
'EDGE_ENHANCE',
'EDGE_ENHANCE_MORE',
'EMBOSS',
'FIND_EDGES',
'SHARPEN',
'SMOOTH',
'SMOOTH_MORE']
```

下一个例子展示的是如何在任意可支持的图像上应用所有当前支持的固定滤波器。

```
import os
import sys
from PIL import Image, ImageChops, ImageFilter
class DemoPIL(object):
    def __init__(self, image_file=None):
        self.fixed_filters = [ff for ff in dir(ImageFilter) if ff.isupper()]
        assert image_file is not None
        assert os.path.isfile(image_file) is True
        self.image_file = image_file
        self.image = Image.open(self.image_file)
```

```
def _make_temp_dir(self):
    from tempfile import mkdtemp
    self.ff_tempdir = mkdtemp(prefix="ff_demo")
def _get_temp_name(self, filter_name):
    name, ext = os.path.splitext(os.path.basename(self.image_file))
    newimage_file = name + "-" + filter_name + ext
    path = os.path.join(self.ff_tempdir, newimage_file)
    return path
def _get_filter(self, filter_name):
    note the use python's eval() builtin here to return function object
    real_filter = eval("ImageFilter." + filter_name)
    return real_filter
def apply_filter(self, filter_name):
    print "Applying filter: " + filter_name
    filter_callable = self._get_filter(filter_name)
    # prevent calling non-fixed filters for now
    if filter_name in self.fixed_filters:
        temp_img = self.image.filter(filter_callable)
    else:
        print "Can't apply non-fixed filter now."
    return temp_img
def run_fixed_filters_demo(self):
    self._make_temp_dir()
    for ffilter in self.fixed_filters:
        temp_img = self.apply_filter(ffilter)
        temp_img.save(self._get_temp_name(ffilter))
    print "Images are in: {0}".format((self.ff_tempdir),)
```

```
if __name__ == "__main__":
    assert len(sys.argv) == 2
    demo_image = sys.argv[1]
    demo = DemoPIL(demo_image)
    # will create set of images in temporary folder
    demo.run_fixed_filters_demo()
```

我们可以从命令行容易地运行改代码：

```
$ python ch06_rec01_01_pil_demo.py image.jpeg
```

把这个示例代码封装在 DemoPIL 类中，这样就易于对它进行扩展，在示例函数run_fixed_filters_demo中共享相同的代码。在这里，相同的代码包括打开图像文件、测试文件是否是一个真实的文件、创建临时目录来存储滤波后的图像、创建滤波后的图像的文件名和向用户打印有用的信息。通过这种方式把代码更好地组织起来，从而能容易地让我们关注在演示函数上，而不用去接触代码的其他部分。

这个示例将打开图像文件，对该图像应用ImageFilter中可用的每一个固定滤波器，并将滤波后的图像存储到一个唯一的临时文件夹中。我们可以得到这个临时文件夹的位置，这样就可以用操作系统的文件管理器打开它并查看所创建的图像。

作为一个可选的练习，你可以尝试扩展这个示例类来向给定的图像应用 ImageFilter中其他可用的滤波器。

6.2.2 操作步骤

本节的例子将演示如何处理某一特定文件夹下的所有图像文件。指定一个目标路径，用程序读取目标路径（图像文件夹）下的所有图像文件，并按给定比例（本例中为0.1）调整它们的大小，然后把每一个文件存储到一个叫做thumbnail_folder的文件夹中。

```
import os
import sys
from PIL import Image
class Thumbnailer(object):
    def __init__(self, src_folder=None):
        self.src_folder = src_folder
        self.ratio = .3
        self.thumbnail_folder = "thumbnails"
    def _create_thumbnails_folder(self):
        thumb_path = os.path.join(self.src_folder, self.thumbnail_folder)
        if not os.path.isdir(thumb_path):
            os.makedirs(thumb_path)
    def _build_thumb_path(self, image_path):
        root = os.path.dirname(image_path)
        name, ext = os.path.splitext(os.path.basename(image_path))
        suffix = ".thumbnail"
        return os.path.join(root, self.thumbnail_folder, name + suffix + ext)
    def _load_files(self):
        files = set()
        for each in os.listdir(self.src_folder):
            each = os.path.abspath(self.src_folder + '/' + each)
            if os.path.isfile(each):
                files.add(each)
        return files
    def _thumb_size(self, size):
        return (int(size[0] * self.ratio), int(size[1] * self.ratio))
    def create_thumbnails(self):
```

```
        self._create_thumbnails_folder()
        files = self._load_files()
        for each in files:
            print "Processing: " + each
            try:
                img = Image.open(each)
                thumb_size = self._thumb_size(img.size)
                resized = img.resize(thumb_size, Image.ANTIALIAS)
                savepath = self._build_thumb_path(each)
                resized.save(savepath)
            except IOError as ex:
                print "Error: " + str(ex)
if __name__ == "__main__":
    # Usage:
    # ch06_rec01_02_pil_thumbnails.py my_images
    assert len(sys.argv) == 2
    src_folder = sys.argv[1]
    if not os.path.isdir(src_folder):
        print "Error: Path '{0}' does not exits.".format((src_folder))
        sys.exit(-1)
    thumbs = Thumbnailer(src_folder)
    # optionally set the name of thumbnail folder inside *src_folder*.
    thumbs.thumbnail_folder = "THUMBS"
    # define ratio to resize image to
    # 0.1 means the original image will be resized to 10% of its size
    thumbs.ratio = 0.1
    # will create set of images in temporary folder
```

```
thumbs.create_thumbnails()
```

6.2.3 工作原理

对于给定的 src_folder 文件夹，我们加载文件夹中的所有文件并尝试用 Image. open()加载其中的每一个文件，这是 create_thumbnails()函数的逻辑。如果尝试加载的文件不是一个图像文件，程序将抛出IOError异常，并打印出错误信息，然后忽略这个文件去顺序地读取下一个文件。

如果想对所加载的文件有更多的控制，应当改变_load_files()函数让它只包括特定扩展名（文件类型）的文件，代码如下。

```
for each in os.listdir(self.src_folder):
    if os.path.isfile(each) and os.path.splitext(each) is in ('.jpg', '.png'):
        self._files.add(each)
```

这并不是安全的做法，因为文件扩展名并没有定义文件类型，它只是帮助操作系统为文件关联了一个默认的程序，但是这种方式在大多数情况下是适用的，并且比读取文件头来确定文件内容（这也不能保证文件就真正是其前几个字节所说的格式）要简单。

6.2.4 补充说明

通过PIL可以容易地把图像从一种格式转换到另一种格式，尽管这种方式使用的不是很多。这通过两个简单的操作就可以做到：首先，使用 open()以原格式打开一幅图像，然后用 save()把图像保存成另一种格式。文件格式可以通过文件名的扩展（ .png 或者.jpeg）隐式地指定，也可以通过传入save()函数的格式参数显式地给出。

6.3 绘制带图像的图表

除了纯数据值之外，图像可以用来增强可视化的效果。很多例子已经证明，通过使用象征性的图像，我们可以把图表更深刻地映射到观察者的心智模型，从而帮助他们更好地更持久地记住可视化的信息。一种做法是在数据上放置图像，把数据值和它们要展示的内容映射起来。matplotlib库可以实现这样的功能，我们将演示如何做到这一点。

6.3.1 准备工作

我们使用Bobby Henderson创作的故事The Gospel of the Flying Spaghetti Monster by Bobby Henderson中一个虚构的例子。在这个故事中，作者把海盗数和海面温度关联起来。为了强调这种关联，我们用测量了海面温度的相同年份的海盗数量按比例显示成海盗船的大小。

我们将利用 Python matplotlib 库的功能，使用可进行高级位置设置的图像和文本，并用箭头对图表进行注解。

所有下一节所需要的文件都可以在ch06文件夹下的源代码库中找到。

6.3.2 操作步骤

下面的例子演示了如何用图像和文本向一幅图表添加注解。

```
import matplotlib.pyplot as plt
from matplotlib._png import read_png
from matplotlib.offsetbox import TextArea, OffsetImage,\
    AnnotationBbox
```

```
def load_data():
    import csv
    with open('pirates_temperature.csv', 'r') as f:
        reader = csv.reader(f)
        header = reader.next()
        datarows = []
        for row in reader:
            datarows.append(row)
    return header, datarows
def format_data(datarows):
    years, temps, pirates = [], [], []
    for each in datarows:
        years.append(each[0])
        temps.append(each[1])
        pirates.append(each[2])
    return years, temps, pirates
```

在定义完helper函数之后，我们可以开始着手创建图表对象，并向其添加子区。我们将把船的图片按比例调整到合适的大小，并用其对每一年的数据进行注解，代码如下。

```
if __name__ == "__main__":
    fig = plt.figure(figsize=(16,8))
    ax = plt.subplot(111) # add sub-plot
header, datarows = load_data()
xlabel, ylabel, _ = header[0]heador[1]
years, temperature, pirates = format_data(datarows)
title = "Global Average Temperature vs. Number of Pirates"
plt.plot(years, temperature, lw=2)
```

```
plt.xlabel(xlabel)
plt.ylabel(ylabel)
# for every data point annotate with image and number
for x in xrange(len(years)):
    # current data coordinate
    xy = years[x], temperature[x]
    # add image
    ax.plot(xy[0], xy[1], "ok")
    # load pirate image
    pirate = read_png('tall-ship.png')
    # zoom coefficient (move image with size)
    zoomc = int(pirates[x]) * (1 / 90000.)
    # create OffsetImage
    imagebox = OffsetImage(pirate, zoom=zoomc)
    # create anotation bbox with image and setup properties
    ab = AnnotationBbox(imagebox, xy,
        xybox=(-200.*zoomc, 200.*zoomc),
        xycoords='data',
        boxcoords="offset points",
        pad=0.1,
        arrowprops=dict(arrowstyle="->",
            connectionstyle="angle,angleA=0,angleB=-30,rad=3")
        )
    ax.add_artist(ab)
    # add text
    no_pirates = TextArea(pirates[x], minimumdescent=False)
    ab = AnnotationBbox(no_pirates, xy,
```

```
        xybox=(50., -25.),
        xycoords='data',
        boxcoords="offset points",
        pad=0.3,
        arrowprops=dict(arrowstyle="->",
            connectionstyle="angle,angleA=0,angleB=-30,rad=3")
        )
    ax.add_artist(ab)
plt.grid(1)
plt.xlim(1800, 2020)
plt.ylim(14, 16)
plt.title(title)
plt.show()
```

上述代码将生成如图6-1所示的图表。

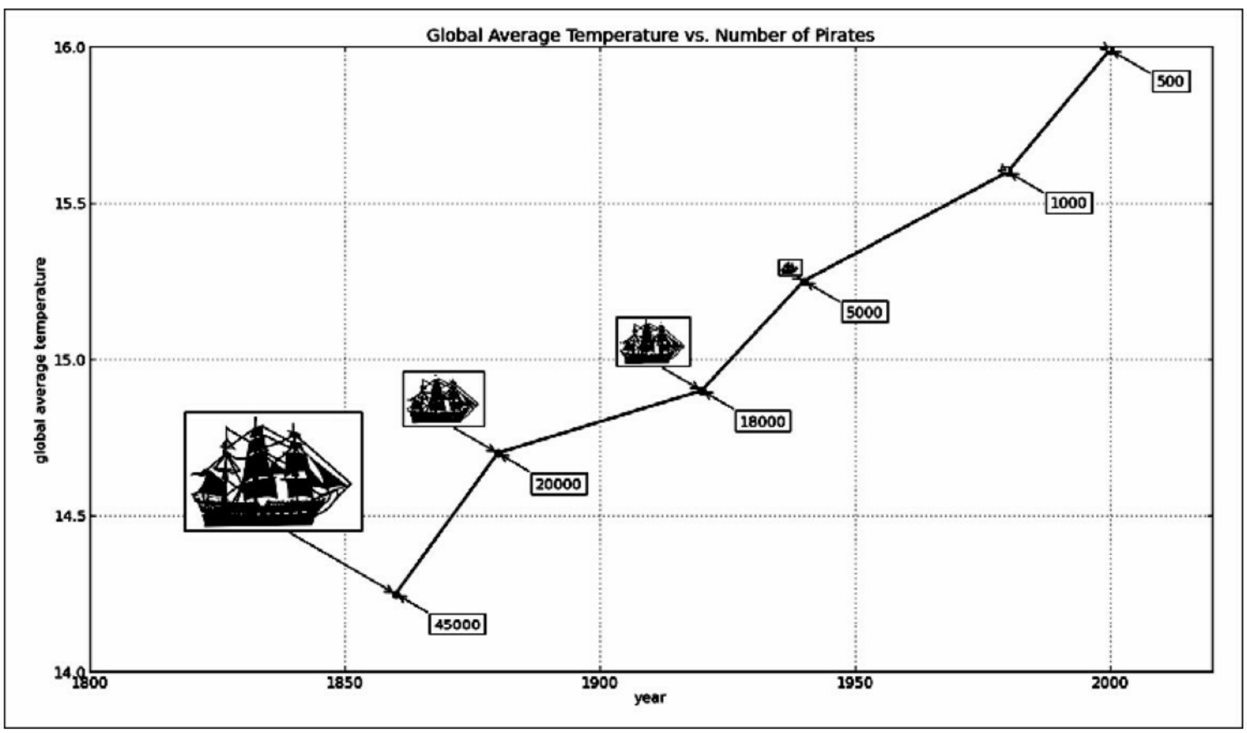


图6-1

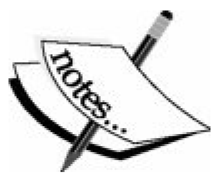
6.3.3 工作原理

我们从创建一个大小合适（也就是16×8）的图表开始。我们需要这个尺寸来适应我们想显示的图像大小。现在我们使用 csv 模块从文件加载数据。实例化一个 csv reader 对象之后，就可以对文件数据进行逐行地迭代了。注意第一行很特殊，它是描述数据列的列头。因为已经在x轴上绘制了年份，在y轴上绘制了温度，读取坐标轴标签值的代码如下。

```
xlabel, ylabel, _ = header
```

并用下面两行代码设置坐标轴标签。

```
plt.xlabel(xlabel)
plt.ylabel(ylabel)
```



在这里，我们使用简洁的 Python 惯例来将文件头解包（unpack）成 3 个变量，当使用“_”作为变量名时，表明我们不关心那个变量的值。

从load_data函数将header和datarows列表返回给调用端main。

通过函数 format_data()读取到列表中的每一个元素，并把每一个单独的实体（年份、温度和海盗数）添加到与该实体相关的ID列表。

年份显示在x轴上，温度显示在y轴上。海盗数显示为一幅海盗船的图片，同时为了提高精度，也将海盗数量显示出来。

用标准的 plot()函数绘制出年份/温度值，除了把线条设置得宽一点（2 pt）之外，没有再添加额外的效果。

然后，为每一个值添加一幅海盗船图片来说明给定年份的海盗数。对此，我们在值的长度范围上（range(len(years))）进行循环，在每一个

年份/温度坐标上画上一个黑点：

```
ax.plot(xy[0], xy[1], "ok")
```

使用helper函数read_png把船的图片从文件加载到一个恰当的数组格式：

```
pirate = read_png('tall-ship.png')
```

然后，计算出缩放系数(zoomc)，以便我们能够按照当前值（pirates[x]）的海盗数按比例调整图像大小，并用相同的系数把图片放置在图表中合适的位置上。

然后，实际的图片在OffsetImage（带有与其父亲AnnotationBbox相关位置的图像容器）中被实例化。

AnnotationBbox是一个像注解一样的类，但是它能显示其他的OffsetBox实例，而不是像Axes.annotate函数那样只显示文本。这允许我们在注解中加载一幅图像或者文本对象，并把它放置在与数据点有一定距离的地方，也可以使用箭头功能（arrowprops）精确地指向一个被注解的数据点。

AnnotateBbox构造函数支持以下参数。

◆ Imagebox：必须是一个OffsetBox 实例（例如OffsetImage），它是注解框的内容。

◆ xy:与注解关联的数据点坐标。

◆ xybox:指定注解框的位置。

◆ xycoords:指定 xy使用的坐标系统（例如数据坐标）。

◆ boxcoords:指定 xybox使用的坐标系统（例如距离 xy位置的偏移）。

◆ pad:指定内边距（padding）的数量。

◆ arrowprops:用于绘制注解边框与数据点的连接箭头的属性字典。

我们使用 pirates 列表中的相同数据项向这个图形添加文本注解，注解的相对位置稍微有些不同。第二个AnnotationBbox的大多数参数和第

一个相同，我们调整了xybox和pad，以便将文本放置在线条的另一边。文本在TextArea类的实例中，这和我们对图像做的事情相似，但是这里的time.TextArea文本和OffsetImage继承自相同的父类OffsetBox。

在TextArea实例中设置文本为no_pirates，并把它放在AnnotationBbox中。

6.4 在具有其他图形的图表中显示图像

本节将演示如何简单但有效地使用 Python matplotlib 库来处理图像通道，并显示外部图像的单通道直方图。

6.4.1 准备工作

虽然我们已经提供了一些样本图像，但是假如图像文件是matplotlib的imread函数所支持的，那么就可以用我们的代码来加载它。

本节将学习如何组合不同的matplotlib图形来实现一个简单的图像查看器的功能。该图像查看器可以显示红、绿、蓝三个通道的图像直方图。

6.4.2 操作步骤

为了演示如何搭建一个图像直方图查看器，我们将实现一个简单的ImageViewer类，该类包含的helper方法的操作如下：

1.加载图像。

2.从图像矩阵中分离出RGB通道。

3.配置图表和坐标轴（子区）。

4.绘制通道直方图。

5.绘制图像。

下面的代码演示了如何创建一个图像直方图查看器。

```
import matplotlib.pyplot as plt
import matplotlib.image as mplimage
import matplotlib as mpl
```

```
import os
class ImageViewer(object):
    def __init__(self, imfile):
        self._load_image(imfile)
        self._configure()
        self.figure = plt.gcf()
        t = "Image: {0}".format(os.path.basename(imfile))
        self.figure.suptitle(t, fontsize=20)
        self.shape = (3, 2)
    def _configure(self):
        mpl.rcParams['font.size'] = 10
        mpl.rcParams['figure.autolayout'] = False
        mpl.rcParams['figure.figsize'] = (9, 6)
        mpl.rcParams['figure.subplot.top'] = .9
    def _load_image(self, imfile):
        self.im = mplimage.imread(imfile)
    @staticmethod
    def _get_chno(ch):
        chmap = {'R': 0, 'G': 1, 'B': 2}
        return chmap.get(ch, -1)
    def show_channel(self, ch):
        bins = 256
        ec = 'none'
        chno = self._get_chno(ch)
        loc = (chno, 1)
        ax = plt.subplot2grid(self.shape, loc)
        ax.hist(self.im[:, :, chno].flatten(), bins, color=ch, ec=ec,\
```

```
            label=ch, alpha=.7)
        ax.set_xlim(0, 255)
        plt.setp(ax.get_xticklabels(), visible=True)
        plt.setp(ax.get_yticklabels(), visible=False)
        plt.setp(ax.get_xticklines(), visible=True)
        plt.setp(ax.get_yticklines(), visible=False)
        plt.legend()
        plt.grid(True, axis='y')
        return ax
    def show(self):
        loc = (0, 0)
        axim = plt.subplot2grid(self.shape, loc, rowspan=3)
        axim.imshow(self.im)
        plt.setp(axim.get_xticklabels(), visible=False)
        plt.setp(axim.get_yticklabels(), visible=False)
        plt.setp(axim.get_xticklines(), visible=False)
        plt.setp(axim.get_yticklines(), visible=False)
        axr = self.show_channel('R')
        axg = self.show_channel('G')
        axb = self.show_channel('B')
        plt.show()
if __name__ == '__main__':
    im = 'images/yellow_flowers.jpg'
    try:
        iv = ImageViewer(im)
        iv.show()
    except Exception as ex:
```

```
        print ex
```

6.4.3 工作原理

从代码末尾开始读，我们看到有硬编码的文件名。通过加载命令行参数，使用sys.argv列表把给定的参数传入到im变量中，可以把这些硬编码的文件名替换掉。

我们实例化了一个带有给定图像文件路径的 ImageViewer类对象。在对象实例化期间，我们试着把图像加载到一个数组，通过rcParams字典配置图表，设置图表大小和标题，并指定对象的方法内部使用的对象字段（self.shape）。

这里主要的方法是 show()，它创建了一个图表的布局，并且把图像数组加载到主子区（左列）中。因为这是一幅真实的图像，没必要使用刻度，因此隐藏掉了所有的刻度和刻度标签。

然后为每一个红、绿和蓝色通道调用show_channel()私有方法。这个方法在右边的列上为每一行也创建了新的子区坐标轴。我们在单独的子区中为每个通道绘制直方图。

此外，我们创建一个小图形，去掉了不必要的x轴刻度，并添加了一个图例，以防我们想要在一个非彩色环境下绘制这个图表。因此，我们可以在这些环境中通过图例分辨出不同的通道。

执行完代码后，将得到如图6-2所示的屏幕截图。

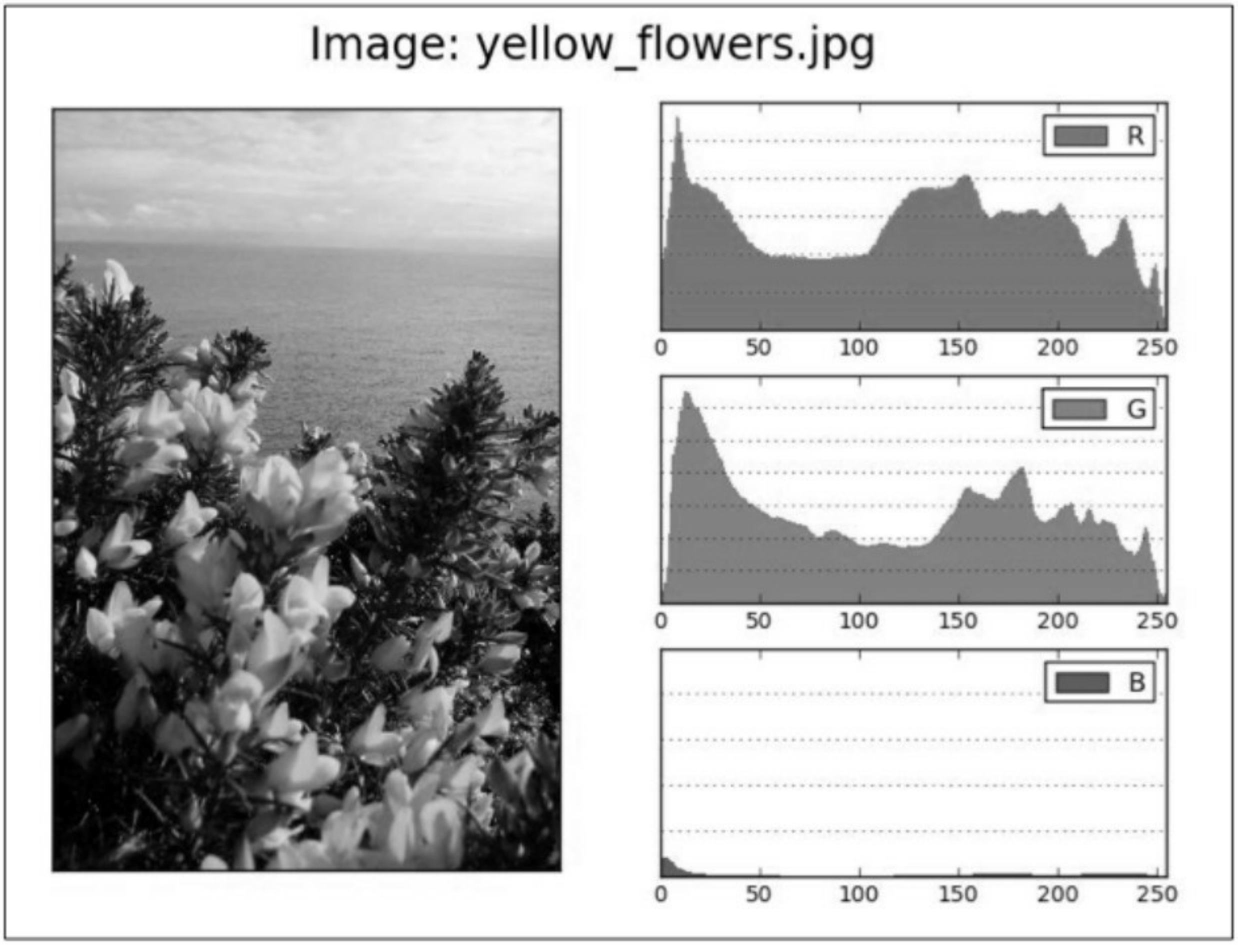


图6-2

6.4.4 补充说明

对于这个图像查看器的例子，使用直方图图表类型仅仅是一种选择。我们可以使用matplotlib支持的任一种图表类型。另一个现实的例子是绘制EEG[1]或者类似的医疗记录，在这种情况下，我们可能想要把切片显示成图像，把所记录的 EEG 的时间序列显示成线形图，并添加关于所显示数据的附加元数据信息，这部分很有可能就是matplotlib.text.Text artists的工作了。

借助于matplotlib与用户GUI事件交互的能力，如果只是手动地缩放一个图形，matplotlib图表也允许我们在想要放大所有图形的地方实现交

互。另一个用法是在显示一幅图像并放大的同时，在当前活跃的图表中也放大其他的图形。一种方法是使用motion_ notify_event调用一个函数更新当前图表中的所有坐标轴（子区）的x轴和y轴范围。

6.5 使用Basemap在地图上绘制数据

或许最好的地理空间可视化是通过把数据叠加在地图上来完成的。无论是整个地球、一个大洲、一个州，甚至是天空，这是让观察者理解其显示的数据与地理关系的一种最简单的方式。

本节将学习如何使用matplotlib的Basemap工具包把数据添加到地图上。

6.5.1 准备工作

既然我们已经熟悉了如何把matplotlib用作绘图引擎，可以继续学习matplotlib其他工具包的功能，例如Basemap地图工具包。

Basemap本身不进行任何绘图的工作，它只是把给定的地理坐标转换到地图投影，并把数据传给matplotlib进行绘图。

首先，我们需要安装Basemap工具包。如果你正在使用EPD，那么Basemap已经被安装好了。如果你是在Linux平台上，最好使用原生的软件包管理器来安装包含Basemap的软件包。例如，在Ubuntu上软件包为python-mpltoolkits.basemap，能通过标准的包管理器进行安装。

$ sudo apt-get install python-mpltoolkits.basemap

在 Mac OS X 上，虽然使用流行的包管理器如 Homebrew、Fink 和 pip 也可以安装，但推荐的方式是使用EPD。

6.5.2 操作步骤

这里有一个例子，关于如何使用Basemap工具包在指定了long、lat坐标对的特定区域绘制简单的墨卡托投影（Mercator projection）：

1.实例化Basemap对象，指定所使用的投影（merc指Mercator）；

2.分别为地图的左下角和右上角指定经度和纬度（在同一个Basemap构造函数中）；

3.创建Basemap地图实例来绘制海岸线和国家；

4.创建Basemap地图实例来填充陆地并绘制地图边界；

5.指示Basemap地图实例绘制子午线和平行线。

下面的代码展示了如何使用Basemap工具包来绘制一个简单的墨卡托投影。

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
map = Basemap(projection='merc',
        resolution = 'h',
        area_thresh = 0.1,
    llcrnrlon=-126.619875, llcrnrlat=31.354158,
    urcrnrlon=-59.647219, urcrnrlat=47.517613)
map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color='coral', lake_color='aqua')
map.drawmapboundary(fill_color='aqua')
map.drawmeridians(np.arange(0, 360, 30))
map.drawparallels(np.arange(-90, 90, 30))
plt.show()
```

这将生成地球的一个可识别区域，如图6-3所示。

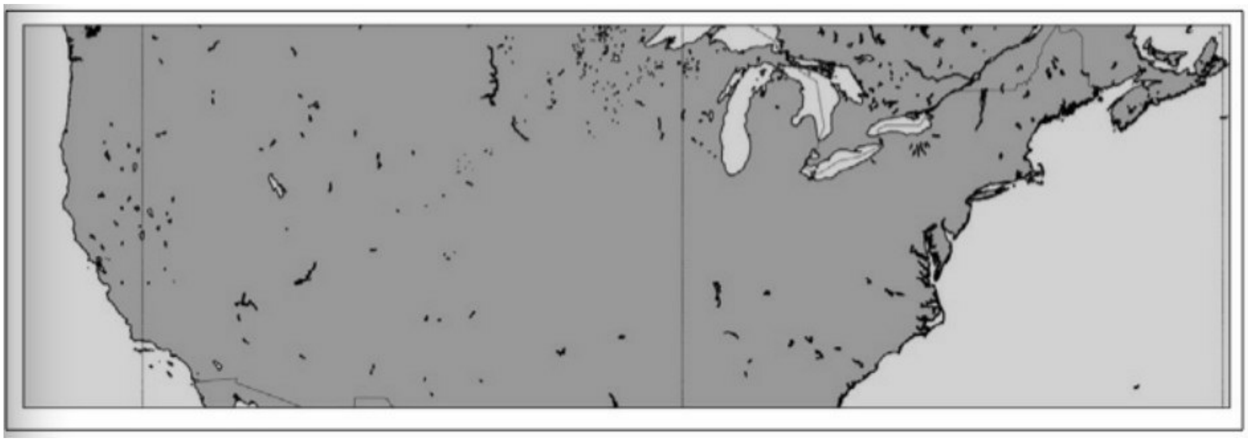


图6-3

既然我们已经知道了如何绘制一幅地图，接着我们需要知道如何在这个地图上绘制数据。如果我们还记得Basemap是一个大的转码器，把经度和纬度对转化到当前地图投影中，那么就明白所有我们需要的是一个包含long/lat的数据集合，并把它传递给Basemap用来投影，然后用matplotlib在地图上把数据绘制出来。我们从cities.shp和cities.shx文件加载美国城市的坐标并把它们投射到地图上。文件在代码库的 ch06 文件夹下，下面是完成这项工作的代码。

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
map = Basemap(projection='merc',
    resolution = 'h',
    area_thresh = 100,
  llcrnrlon=-126.619875, llcrnrlat=25,
  urcrnrlon=-59.647219, urcrnrlat=55)
shapeinfo = map.readshapefile('cities','cities')
x, y = zip(*map.cities)
# build a list of US cities
city_names = []
```

```
for each in map.cities_info:
    if each['COUNTRY'] != 'US':
        city_names.append("")
    else:
        city_names.append(each['NAME'])
map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color='coral', lake_color='aqua')
map.drawmapboundary(fill_color='aqua')
map.drawmeridians(np.arange(0, 360, 30))
map.drawparallels(np.arange(-90, 90, 30))
# draw city markers
map.scatter(x,y,25, marker='o',zorder=10)
# plot labels at City coords.
for city_label, city_x, city_y in zip(city_names, x, y):
    plt.text(city_x, city_y, city_label)
plt.title('Cities in USA')
plt.show()
```

6.5.3 工作原理

Basemap用法的基本原理是，导入主要的模块和实例化一个带有期望属性的Basemap类。在实例化阶段必须指定所使用的投影和想处理的地球区域。

在绘制地图和用matplotlib.pyplot.show()显示绘图窗口之前可以应用额外的配置。

Basemap支持很多（精确地说是 32 个）不同的投影。其中大多数应

用范围非常窄，但是还有一些是比较通用的，被应用在大多数常见的地图可视化中。



通过查询Basemap模块，我们可以很容易地知道可以使用哪些投影：

```
In [5]: import mpl_toolkits.basemap
In [6]: print mpl_toolkits.basemap.
projections
mbtfpq       McBryde-Thomas Flat-Polar Quartic
aeqd         Azimuthal Equidistant
sinu         Sinusoidal
poly         Polyconic
omerc        Oblique Mercator
gnom         Gnomonic
moll         Mollweide
lcc          Lambert Conformal
tmerc        Transverse Mercator
nplaea       North-Polar Lambert Azimuthal
gall         Gall Stereographic Cylindrical
North-Polar Azimuthal Equidistantnpaeqd
mill         Miller Cylindrical
merc         Mercator
stere        Stereographic
eqdc         Equidistant Conic
cyl          Cylindrical Equidistant
```

npstere North-Polar Stereographic
spstere South-Polar Stereographic
hammer Hammer
geos Geostationary
nsper Near-Sided Perspective
eck4 Eckert IV
aea Albers Equal Area
kav7 Kavrayskiy VII
spaeqd South-Polar Azimuthal Equidistant
ortho Orthographic
cass Cassini-Soldner
vandg van der Grinten
laea Lambert Azimuthal Equal Area
splaea South-Polar Lambert Azimuthal
robin Robinson

通常，我们将绘制整个投影，如果没有特别指定，默认会使用一些合理的值。

在放大地图上的特定区域时，我们会指定要显示的区域的左下角和右上角的经度和纬度。对于这个例子，我们使用墨卡托投影。



在这里我们可以看到缩写的参数名字的描述。

- ◆ llcrnrlon：左下角的经度。
- ◆ llcrnrlat：左下角的纬度。
- ◆ urcrnrlon：右上角的经度。
- ◆ urcrnrlat：右上角的纬度。

6.5.4 补充说明

我们仅仅了解了Basemap工具包功能的皮毛，在官方文档中可以找到更多的例子，地址为http://matplotlib.org/basemap/users/examples.html。

官方 Basemap 文档的例子使用的大多数数据位于远程的服务器上，并且有特定的格式。为了高效地获取这些数据，可以使用 NetCDF 数据格式。NetCDF 是一种常见的数据格式，其设计之初考虑了网络的效率。即使整个数据集合非常大，它也允许程序获取其所需的数据，所以使用这种格式是非常实用的。不需要在每次需要数据和每次数据变化时，下载海量的数据集合并把其存储在本地。

6.6 使用Google Map API在地图上绘制数据

在本节中，我们将脱离桌面环境来演示一下如何把图表输出到 Web 页面。虽然 Web前端使用的主要语言不是Python而是HTML、CSS和JavaScript，但是仍然可以用Python做重要的工作：获取数据、处理数据、执行密集的运算，以及把数据渲染成适用于Web输出的格式，即使用要求的JavaScript版本创建HTML页面来完成可视化工作。

6.6.1 准备工作

我们将使用用于 Python 的 Google 数据可视化库来为前台界面准备数据，并使用另一个Google可视化API在要求的可视化平台，也就是在地图和表格中渲染数据。

开始之前，需要安装 google-visuallization-python 模块。从https://code.google.com/p/google-visualization-python/downloads/detail?name=gviz_api_py-1.8.2.tar.g z&can=2&q=下载最新的稳定版本，解压压缩包并安装模块。操作步骤如下。

```
$ tar xfv gviz_api_py-1.8.2.tar.gz
$ cd gviz_api_py
$ sudo python ./setup.py install
```

在 Windows 和 Mac OS X 平台上需使用合适的软件解压 tar.gz压缩包，其他步骤相同。注意，为了在系统中安装这个模块，我们必须成为超级用户[2]（也就是获得管理员权限）。

如果不想污染你的操作系统包，一个更好的选择是可以仅为本节创建一个virtualenv环境来安装这些包。我们已经在第1章“准备你的工作环

境”中解释了如何处理virtualenv环境。

对于前端库我们不需要安装任何东西，因为这些库会在Web页面中直接从Google服务器上加载。

我们需要为本节激活网络访问，因为输出是一个Web页面，它将在一个浏览器中打开，直接从远程的服务器获取JavaScript库。

本节将学习如何结合使用用于Python的Google数据可视化库和JavaScript来创建Web可视化页面。

6.6.2 操作步骤

下述代码演示了如何使用Python和gdata_viz模块从一个CSV文件加载数据，并使用 Google Geochart 和 Table Visualization 在世界地图上可视化各国的可支配月平均工资。

1.实现一个函数用作模板生成器。

2.使用csv模块从本地CSV文件加载数据。

3.使用DataTable来描述数据，并使用LoadData从Python字典中加载数据。

4.把输出渲染到Web页面。

下面是实现代码：

```
import csv
import gviz_api
def get_page_template():
    page_template = """
    <html>
    <script  src="https://www.google.com/jsapi"  type="text/javascript">
  </script>
    <script>
```

```
    google.load('visualization', '1', {packages:['geochart','table']});
    google.setOnLoadCallback(drawMap);
    function drawMap() {
        var json_data = new google.visualization.DataTable(%s,0.6);
        var options = {colorAxis: {colors: ['#eee', 'green']}};
        var mymap = new google.visualization.GeoChart(
            document.getElementById('map_div'));
        mymap.draw(json_data, options);
        var mytable = new google.visualization.Table(
            document.getElementById('table_div'));
        mytable.draw(json_data, {showRowNumber: true})
    }
    </script>
    <body>
    <H1>Median Monthly Disposable Salary World Countries</H1>
    <div id="map_div"></div>
    <hr />
    <div id="table_div"></div>
    <div id="source">
    <hr />
    <small>
    Source:
    <a href="http://www.numbeo.com/cost-of-living/prices_by_
country.jsp? displayCurrency=EUR&itemId=105">
    http://www.numbeo.com/cost-of-living/prices_by_country.jsp?dis
play Currency=EUR&itemId=105
    </a>
```

```
    </small>
    </div>
    </body>
    </html>
    """
    return page_template
def main():
    # Load data from CVS file
    afile = "median-dpi-countries.csv"
    datarows = []
    with open(afile, 'r') as f:
        reader = csv.reader(f)
        reader.next() # skip header
        for row in reader:
            datarows.append(row)
```



```
    # Describe data
    description = {"country": ("string", "Country"),
            "dpi": ("number", "EUR"), }
    # Build list of dictionaries from loaded data
    data = []
    for each in datarows:
        data.append({"country": each[0],
            "dpi": (float(each[1]), each[1])})
    # Instantiate DataTable with structure defined in 'description'
    data_table = gviz_api.DataTable(description)
    # Load it into gviz_api.DataTable
```

```
    data_table.LoadData(data)
    # Creating a JSon string
    json = data_table.ToJSon(columns_order=("country", "dpi"),
        order_by="country", )
    # Put JSON string into the template
    # and save to output.html
    with open('output.html', 'w') as out:
        out.write(get_page_template() % (json,))
if __name__ == '__main__':
    main()
```

这段代码将生成output.html文件，可以用我们喜爱的Web浏览器打开它。页面看上如图6-4所示。



图6-4

6.6.3 工作原理

这段代码的主入口点是main()函数。首先，使用csv模块加载数据。我们可以从公共网站www.numbeo.com获得该数据，并把它存储为CSV格式。最终的文件可以从代码库中本章的ch06文件夹中获得。为了能够使用Google数据可视化库，我们需要把数据描述给它。这里我们可以用Python字典描述数据，指定列ID、数据类型和可选的标签。在接下来的例子中数据定义成以下格式。

{"name": ("data_type", "Label")}:

```
description = {"country": ("string", "Country"),
   "dpi": ("number", "EUR"), }
```

然后，需要把加载的CSV数据行映射到这个格式上。我们将在data变量中创建一个字典的列表。

现在，我们具备了用所描述的数据结构的gviz_data.DataTable实例化data_table的所有内容。接下来，把数据加载到其中并以JSON格式输出到page_template中。

get_page_template()函数包含了这段逻辑的剩余部分。它包含了生成HTML页面的一段客户端（前端）代码，以及从 Google 服务器加载Google 数据可视化库的一段JavaScript 代码。加载 Google 的 JavaScript API 的代码行如下。

```
<script src="https://www.google.com/jsapi"
   type="text/javascript"></script>
```

跟随其后的是另一对<script>...</script>标签，其包含了一个额外的设置。首先，我们加载Google数据可视化库和所需的包——geochart和table。

```
google.load('visualization', '1', {packages:['geochart','table']});
```

然后，我们设置了一个函数，该函数在页面加载时会被调用。在Web世界中该事件被注册为onLoad，因此回调函数通过setOnLoadCallback函数进行设置。

```
google.setOnLoadCallback(drawMap);
```

这里的含义是：当页面加载时，google实例将调用我们定义的自定义函数drawMap()。drawMap函数把一个JSON字符串加载到DataTable实例的JavaScript版本中。

```
var json_data = new google.visualization.DataTable(%s, 0.6);
```

接下来，在ID为map_div的HTML元素中创建了一个geochart实例。

```
var mymap = new google.visualization.GeoChart(
```

```
   document.getElementById('map_div'));
```

用json_data绘制地图，并且提供自定义的options。

```
mymap.draw(json_data, options);
```

类似地，在地图下面渲染出Google的JavaScript表。

```
var mytable = new google.visualization.Table(
   document.getElementById('table_div'));
mytable.draw(json_data, {showRowNumber: true})
```

把输出保存为HTML文件，这样就可以在浏览器中打开它。这对于一个Web服务的动态渲染用处不大。有一个更好的方式是，从Python代码中直接输出HTTP应答，然后创建一个后台服务来响应客户端的Web请求，返回客户端可以加载和渲染的JSON数据。



如果想了解关于如何读取HTTP应答的知识，请登录网址http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Response_message 阅读更多关于 HTTP协议和应答消息的内容。

我们通过把ToJson()调用替换成有相同签名的ToJsonResponse()可以做到这一点。这个调用将返回一个包含payload（被JavaScript客户端消费的JSON化的data_table）的HTTP应答。

6.6.4 补充说明

当然，这只是一个例子，演示了如何把Python作为一种后台语言，在服务器上执行数据获取和处理的工作，同时把前台的工作留给通用的HTML/JavaScript/CSS等一系列语言。这让我们能向广泛的受众提供可视化的交互式和动态的界面，而不需要他们安装任何东西（好吧，除了Web浏览器之外，但是这通常在电脑或者智能手机中已经安装了）。说

到这里，我们一定注意到这些输出的质量不像matplotlib输出的质量那么高，而高质量的输出正是matplotlib的强项。

为了用Web（和Python）做更多的工作，必须学习更多关于Web的知识和其使用的语言。本书不会涵盖这些内容，但是会对如何使用知名的第三方库生成满意的Web输出，同时尽可能少地编写Web代码。生成一个可能的解决方案，给出了一些思路。

更多的文档可以从 Google 开发者门户网站获得，网址为https://developers. google.com/chart/interactive/docs/dev/gviz_api_lib。

6.7 生成CAPTCHA图像

虽然这不是通常所指的严格意义上的数据可视化，但是用Python生成图像的能力在很多情况下都非常有用，这就是其中之一。

本节将介绍如何生成用来区分人类和电脑的随机图像——CAPTCHA[3]图像。

6.7.1 准备工作

CAPTCHA 是指全自动区分计算机和人类的图灵测试（Completely Automated Public Turing test to tell Computers and Humans Apart），由卡耐基梅隆大学注册商标。这个测试被用来挑战自动填充各种 Web 表单的计算机程序（通常指机器人），这些表单主要是针对人类的，不应该被自动化。常见的例子有注册表单、登录表单、调查表等。

CAPTCHA 本身可以有很多形式，但是最常见的形式是人类应该能够读取一幅带有扭曲的字符和数字的图像，并在相应的字段中填入结果。

本节将学习如何利用Python的图像库来生成图像、渲染点和线，以及渲染文本。

6.7.2 操作步骤

通过执行下面的步骤，我们将演示在创建一个简单的个人CAPTCHA生成器时所涉及的内容。

1.设置图像大小、文本、字体大小、背景颜色和CAPTCHA长度。

2.从英文字母表中选取随机的字符。

3.用指定的字体和颜色在图像中把这些字符绘制出来。

4.添加一些直线和弧线形式的噪声。

5.把CAPTCHA和图像对象返回给调用者。

6.把生成的图像显示给用户。

下面的代码演示了如何生成简单的个人CAPTCHA生成器。

```
from PIL import Image, ImageDraw, ImageFont
import random
import string
class SimpleCaptchaException(Exception):
    pass
class SimpleCaptcha(object):
    def __init__(self, length=5, size=(200, 100), fontsize=36,
            random_text=None, random_bgcolor=None):
        self.size = size
        self.text = "CAPTCHA"
        self.fontsize = fontsize
        self.bgcolor = 255
        self.length = length
        self.image = None # current captcha image
        if random_text:
            self.text = self._random_text()
        if not self.text:
raise SimpleCaptchaException("Field text must not beempty.")
    if not self.size:
        raise SimpleCaptchaException("Size must not be empty.")
    if not self.fontsize:
        raise SimpleCaptchaException("Font size must be defined.")
```

```
    if random_bgcolor:
        self.bgcolor = self._random_color()
    def _center_coords(self, draw, font):
        width, height = draw.textsize(self.text, font)
        xy = (self.size[0] - width) / 2., (self.size[1] - height) / 2.
        return xy
    def _add_noise_dots(self, draw):
        size = self.image.size
        for _ in range(int(size[0] * size[1] * 0.1)):
            draw.point((random.randint(0, size[0]),
                random.randint(0, size[1])),
                fill="white")
        return draw
    def _add_noise_lines(self, draw):
        size = self.image.size
    for _ in range(8):
        width = random.randint(1, 2)
        start = (0, random.randint(0, size[1] - 1))
        end = (size[0], random.randint(0, size[1] - 1))
        draw.line([start, end], fill="white", width=width)
    for _ in range(8):
        start = (-50, -50)
        end = (size[0] + 10, random.randint(0, size[1] + 10))
        draw.arc(start + end, 0, 360, fill="white")
    return draw
def get_captcha(self, size=None, text=None, bgcolor=None):
    if text is not None:
```

```
            self.text = text
        if size is not None:
            self.size = size
        if bgcolor is not None:
            self.bgcolor = bgcolor
        self.image = Image.new('RGB', self.size, self.bgcolor)
        # Note that the font file must be present
        # or point to your OS's system font
        # Ex. on Mac the path should be '/Library/Fonts/Tahoma.ttf'
        font = ImageFont.truetype('fonts/Vera.ttf', self.fontsize)
        draw = ImageDraw.Draw(self.image)
        xy = self._center_coords(draw, font)
        draw.text(xy=xy, text=self.text, font=font)
        # Add some dot noise
        draw = self._add_noise_dots(draw)
        # Add some random lines
        draw = self._add_noise_lines(draw)
        self.image.show()
        return self.image, self.text
    def _random_text(self):
        letters = string.ascii_lowercase + string.ascii_uppercase
        random_text = ""
        for _ in range(self.length):
            random_text += random.choice(letters)
        return random_text
    def _random_color(self):
        r = random.randint(0, 255)
```

```
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    return (r, g, b)
if __name__ == "__main__":
    sc = SimpleCaptcha(length=7, fontsize=36, random_text=True,
random_ bgcolor=True)
    sc.get_captcha()
```

这段代码生成类似图6-5所示的图像。

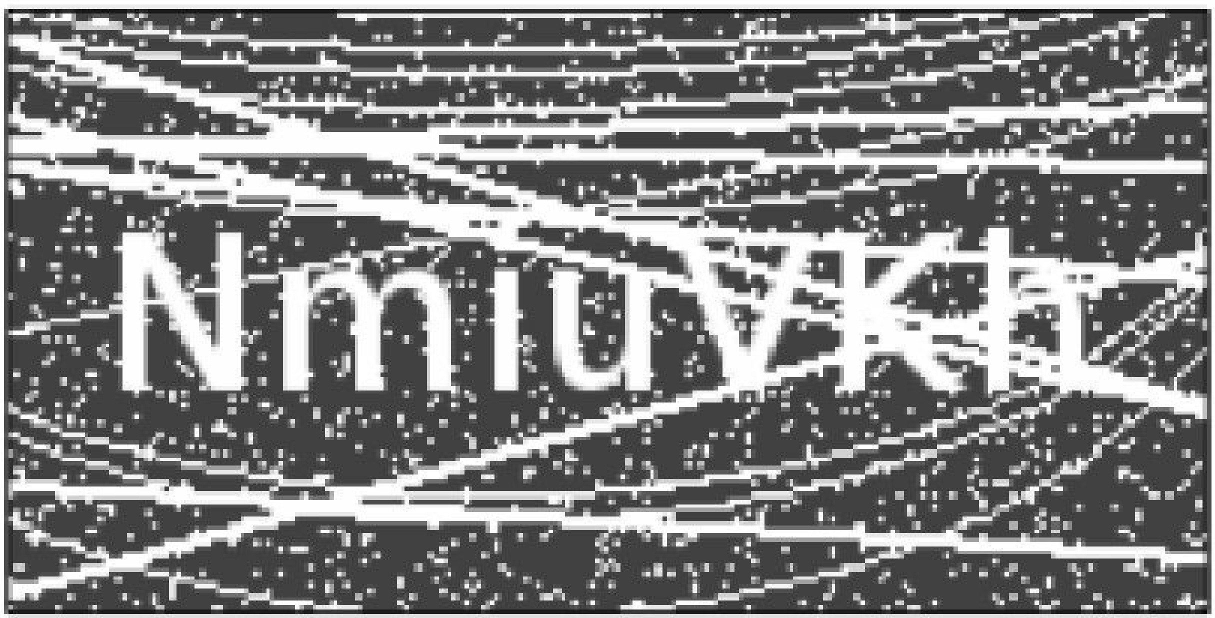


图6-5

6.7.3 工作原理

这个例子描述了如何使用 Python 图像库生成预定义图像，创建一个简单但有效的CAPTCHA生成器的过程。

我们把功能封装到一个类SimpleCaptcha中，因为这为进一步开发提供了一个安全的空间。而且，我们创建了一个自定义的SimpleCaptchaException类来容纳将来的异常类型。



如果你不是在写小段的粗制滥造的脚本，而是开始为你的代码域编写和设计自定义异常类型，不使用原生的Python标准异常通常是件好事。你将会在代码可读性和软件可维护性上获益不少。

从代码尾部的main函数代码段开始看，我们把要生成图像的设置作为参数传给构造函数，实例化类对象。接着，在sc对象上调用get_captcha方法。作为本节演示的目的，get_captcha显示图像对象作为结果，但是我们也可以把图像对象返回给这个方法可能的调用者以供其使用。用法有很多种，调用者可以把图像存储到文件中；或者如果是一个Web应用，可以返回图像流并把结果写到请求该CAPTCHA的客户端。

要注意的一件重要的事情是，为了完成CAPTCHA测试的挑战—应答过程，必须返回在图像上生成的CAPTCHA字符串的文本，这样调用者才可以将用户的应答和期望的值进行比较。

如果用户提供了自定义值，为了覆盖类的默认值，get_captcha 方法首先验证输入的参数。之后，通过Image.new实例化一个新的图像对象。该对象被存储到self.image中，我们用它来绘制和写入文本。在把文本写入图像之后，我们添加了随机放置的点和线，以及一些弧线段的噪声。

这些工作通过_add_noise_points 和_add_noise_lines 完成。第一个函数循环地把一个点添加到图像上的一个不太靠近图像边缘的随机位置，第二个函数从图像的左手边向图像的右手边绘制了几条线段。

6.7.4 补充说明

我们基于一些关于其用法的假设创建了这个类。假设用户只是想接

受默认设置（也就是随机背景颜色上的7个随机字符），然后从其得到结果。这是我们在构造函数上放置helper函数来设置随机文本和随机背景颜色的原因。如果最常见并有效的用法总是覆盖默认设置，那样我们就会想着把这些操作从构造函数中去掉，并放到一个单独的函数调用中。

例如，也许用户总想使用英文单词作为CAPTCHA挑战。如果是这种情况，我们会希望能够只是简单地调用一个方法就可以提供那样的结果。可以创建一个 get_english_captcha 方法，其中包含了构造函数中的随机逻辑，然后从给定的英文字典中挑选随机单词。Unix 系统上，在/usr/share/dict/words 中有一个常用的英文字典，我们可以用它来完成这件事。代码如下。

```
def get_english_captcha(self):
    words = '/usr/share/dict/words'
    with open(words, 'r') as wf:
        words = wf.readlines()
        aword = random.choice(words)
        aword = aword.strip() # remove newline and spaces
    return self.get_captcha(text=aword)
```

总的来说，生成CAPTCHA的例子没达到产品级质量，因此必须在使用前添加更多的保护和随机性如字符旋转。

如果需要保护你的Web表单来防止机器人的攻击，应当重用一些已有的第三方Python模块和库。甚至已经有专门为现有的Web框架创建的模块。

甚至有一些Web服务，如带有经过验证的Python模块recaptcha-client（https://pypi.python. org/pypi/recaptcha-client）的reCAPTCHA（http://www.google.com/recaptcha），注册之后就可以使用了。它不需要任何图像库，因为图像直接从reCAPTCHAWeb 服务获

取，但是它有其他一些依赖如pycrypto。通过使用这个Web服务和库，你同时也在为使用通用字符识别（OCR）技术从Google图书项目或者旧版纽约时报的图书扫描工作做贡献。从reCAPTCHA网站你可以获得更多内容。

注释

[1]. EEG: electroencephalo-graph, 脑电图

[2]. Windows 下没有 sudo 命令，保证运行命令的当前用户有管理员权限即可。

[3]. CAPTCHA：俗称验证码，在本书中采用缩写，不做翻译。

第7章 使用正确的图表理解数据

本章将学习以下内容。

◆ 理解对数图

◆ 理解频谱图

◆ 创建火柴杆图

◆ 绘制矢量场流线图

◆ 使用颜色表

◆ 使用散点图和直方图

◆ 绘制两个变量间的互相关图形

◆ 自相关的重要性

7.1 简介

在本章中，我们将更多地关注用我们展示的数据来理解我们想要表达什么，以及如何有效地把它表达出来。我们将展示一些新的技术和图表，但是所有这些都将通过对我们想要传达给用户的信息的理解而得到增强。让我们问一个这样的问题，“为什么要以这种方式展示数据？”这在数据探索阶段是一个最重要的问题。如果没能很好地理解数据而把它以某种形式展示出来，那么毫无疑问，读者将无法正确地理解这些数据。

7.2 理解对数图

通常情况下，在读日报及类似的文章时，人们就能发现媒体机构用图表歪曲了事实。一个常见的例子是用线性标度来创建所谓的恐慌图。图表中有一个在很长一段时间（若干年）内持续增长的值，其起始值要比最新的值小好几个量级。然而在正确的可视化时，这些值可以（并且通常应该）用线形图或者近似线性的图表表示，把它们要强调的一些恐慌从文章中去掉。

7.2.1 准备工作

使用对数标度时，连续值的比例是常量。这在读对数图表时是非常重要的。使用线性（算术）标度时，常量是连续值之间的距离。换句话说，对数图表按数量级顺序有一个常量的距离。这在接下来的图表中可以看到，用于生成图表的代码在后面也会进行解释。

根据一般经验，遇到以下情况应该使用对数标度。

◆ 当要展示的数据的值跨越好几个量级时。

◆ 当要展示的数据有朝向大值（一些数据点比其他数据大很多）的倾斜度时。

◆ 当要展示变化率（增长率），而不是值的变化时。

不要盲目地遵循这些规则，它们更像是指导，而不是规则，要始终依靠你自己对于手头数据和项目，或者客户对你提出的需求作判断。

根据数据范围的不同，应该使用不同的对数底。对数的标准底是10，但是如果数据范围比较小，以2为底数会更有帮助，因为其会在一个较小的数据范围下有更多的分辨率。

如果有适合在对数标度上显示的数据范围，我们会注意到以前非常靠近而难以判断差异的值现在很好地分离开了。相比于在线性标度下展示数据，对数展示让我们读起图来更容易。

对于收集了很长时间序列范围的数据的增长率图表，我们想看的不是在时间点所测量的绝对值，而是在时间上的增长。我们仍可以得到绝对值信息，但是这些信息有较低的优先级。

而且，如果数据分布存在一个正偏态，例如工资，取值（工资）的对数能让数据更合乎模型，因为对数变换能提供一个更加正常的数据分布。

7.2.2 操作步骤

我们将用一段代码来证明上面所述的内容。这段代码用不同的标度（线性和对数）在两个不同的图表中显示了两个相同的数据集合（一个线性的，一个对数的）。

我们将借助后面的代码执行下面的步骤。

◆ 生成两个简单的数据集合：指数/对数 y 和线性 z。

◆ 创建一个包含四个子区的图形。

◆ 创建两个包含数据集合 y 的子区：一个为对数标度，一个为线性标度。

◆ 创建两个包含数据集合 z 的子区：一个为对数标度，一个为线性标度。

代码如下：

```
from matplotlib import pyplot as plt
import numpy as np
x = np.linspace(1, 10)
y = [10 ** el for el in x]
```

```
z = [2 * el for el in x]
fig = plt.figure(figsize=(10, 8))
ax1 = fig.add_subplot(2, 2, 1)
ax1.plot(x, y, color='blue')
ax1.set_yscale('log')
ax1.set_title(r'Logarithmic plot of $ {10}^{x} $ ')
ax1.set_ylabel(r'$ {y} = {10}^{x} $')
plt.grid(b=True, which='both', axis='both')
ax2 = fig.add_subplot(2, 2, 2)
ax2.plot(x, y, color='red')
ax2.set_yscale('linear')
ax2.set_title(r'Linear plot of $ {10}^{x} $ ')
ax2.set_ylabel(r'$ {y} = {10}^{x} $')
plt.grid(b=True, which='both', axis='both')
ax3 = fig.add_subplot(2, 2, 3)
ax3.plot(x, z, color='green')
ax3.set_yscale('log')
ax3.set_title(r'Logarithmic plot of $ {2}*{x} $ ')
ax3.set_ylabel(r'$ {y} = {2}*{x} $')
plt.grid(b=True, which='both', axis='both')
ax4 = fig.add_subplot(2, 2, 4)
ax4.plot(x, z, color='magenta')
ax4.set_yscale('linear')
ax4.set_title(r'Linear plot of $ {2}*{x} $ ')
ax4.set_ylabel(r'$ {y} = {2}*{x} $')
plt.grid(b=True, which='both', axis='both')
```

代码将生成如图7-1所示的图表。

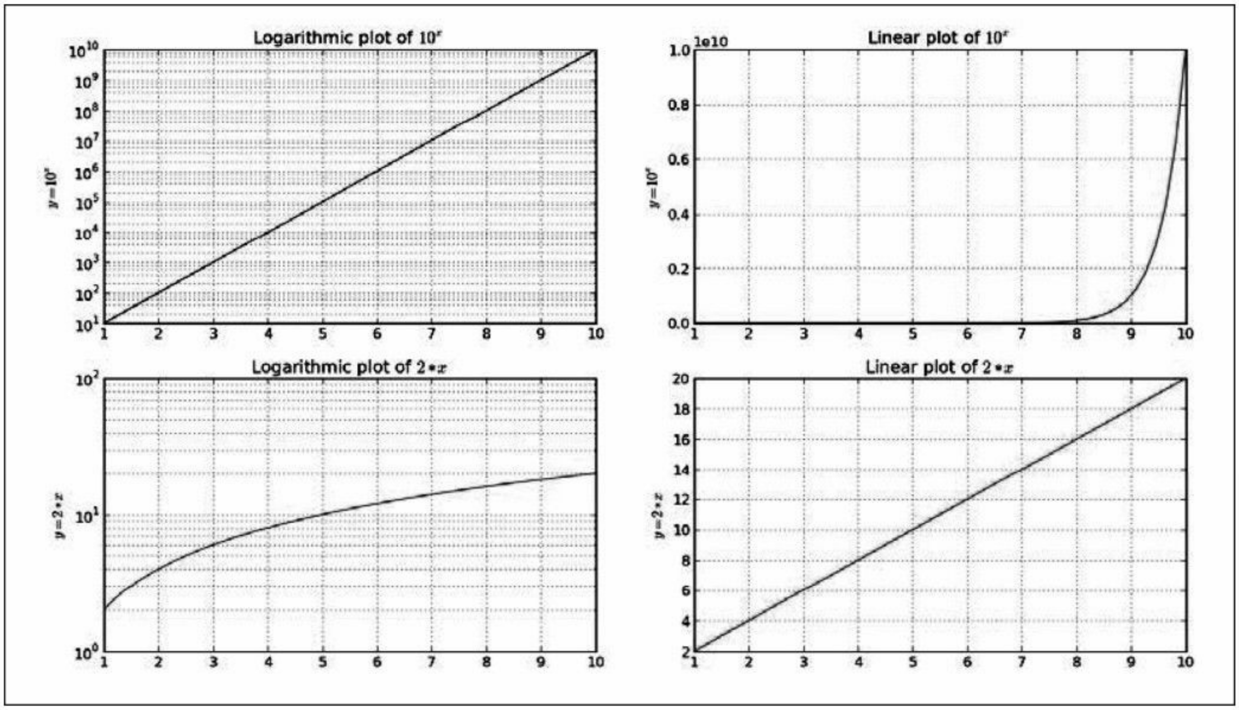


图7-1

7.2.3 工作原理

我们生成了一些样本数据和两个相关的变量：y和z。变量y被表示为x（数据）的指数函数，变量z是x的简单线性函数。这帮助我们展示了线性图表和指数图表的不同。

然后，创建四个子区，上面一行子区是关于数据（x，y）的，下面一行子区是关于数据（x，z）的。

从左手边看，y　轴列为对数标度；从右手边看，y　轴列为线性标度。通过set_yscale('log')分别对每一个坐标轴进行设置。

我们为每一个子区设置了标题和标签，其中标签描述了所绘制的函数。

通过 plt.grid(b=True，which='both'，axis='both')，我们为所有两个坐标轴和主次刻度打开网格显示。

我们观察到，在线性图表中线性函数是直线，在对数图表中对数函

数同样也是直线。

7.3 理解频谱图

频谱图是一个随时间变化的谱表现，它显示了信号的频谱强度随时间的变化。

频谱图是把声音或者其他信号的频谱以可视化的方式呈现出来。它被用在很多科学领域中，从声音指纹如声音识别，到雷达工程学和地震学。

通常，频谱图的布局如下：x轴表示时间，y轴表示频率，第三个维度是频率—时间对的幅值，通过颜色表示。因为这是三维的数据，因此我们也可以创建 3D 图表来表示，其中强度表示为 z 轴上的高度。3D 图表的问题是人们不太容易理解以及进行比较，而且比2D图表占用更多的空间。

7.3.1 准备工作

对于严谨的信号处理，我们将会研究更低级别的细节，进而能从中发现模式以及自动识别一定的特征；但是对于本节数据可视化的内容，我们将借助一些著名的Python库来读取一个音频文件，对它进行采样，然后绘制出频谱图。

为了能读取 WAV 文件并把声音可视化出来，需要做一些准备工作。我们需要安装libsndfile1 系统库来读/写音频文件。这可以通过你喜欢的包管理工具完成。对于Ubuntu，使用以下命令。

$ sudo apt-get install libasound1-dev

安装dev包非常重要，它包含了头文件，从而使pip可以创建scikits.audiolab模块。

我们也可以安装 libasound和 ALSA（Advanced Linux Sound Architecture，高级Linux声音体系）头来避免编译时警告。这是可选的，因为我们不打算使用ALSA库提供的特性。对于 Ubuntu Linux，执行以下命令：

```
$ sudo apt-get install libasound2-dev
```

我们用pip安装用来读取WAV文件的scikits.audiolab：

```
$ pip install scikits.audiolab
```



永远记住要进入当前工程的虚拟环境，因为这样才不会弄脏你的系统库。

7.3.2 操作步骤

本节将使用预录制的声音文件 test.wav，该文件可以在本书的文件代码库中找到，但也可以自己生成一个样本文件。

在这个例子中，我们顺序地执行下面的步骤。

1.读取包含一个已经录制的声音样本的WAV文件。

2.通过NFFT设置用于傅里叶变换的窗口长度。

3.在采样时，使用noverlap设置重叠的数据点。

```
import os
from math import floor, log
from scikits.audiolab import Sndfile
import numpy as np
from matplotlib import pyplot as plt
# Load the sound file in Sndfile instance
soundfile = Sndfile("test.wav")
```

```
# define start/stop seconds and compute start/stop frames
start_sec = 0
stop_sec = 5
start_frame = start_sec * soundfile.samplerate
stop_frame = stop_sec * soundfile.samplerate
# go to the start frame of the sound object
soundfile.seek(start_frame)
# read number of frames from start to stop
delta_frames = stop_frame - start_frame
sample = soundfile.read_frames(delta_frames)
map = 'CMRmap'
fig = plt.figure(figsize=(10, 6), )
ax = fig.add_subplot(111)
# define number of data points for FT
NFFT = 128
# define number of data points to overlap for each block
noverlap = 65
pxx, freq, t, cax = ax.specgram(sample, Fs=soundfile.samplerate,
    NFFT=NFFT, noverlap=noverlap,
    cmap=plt.get_cmap(map))
plt.colorbar(cax)
plt.xlabel("Times [sec]")
plt.ylabel("Frequency [Hz]")
plt.show()
```

代码生成的频谱图如图7-2所示。

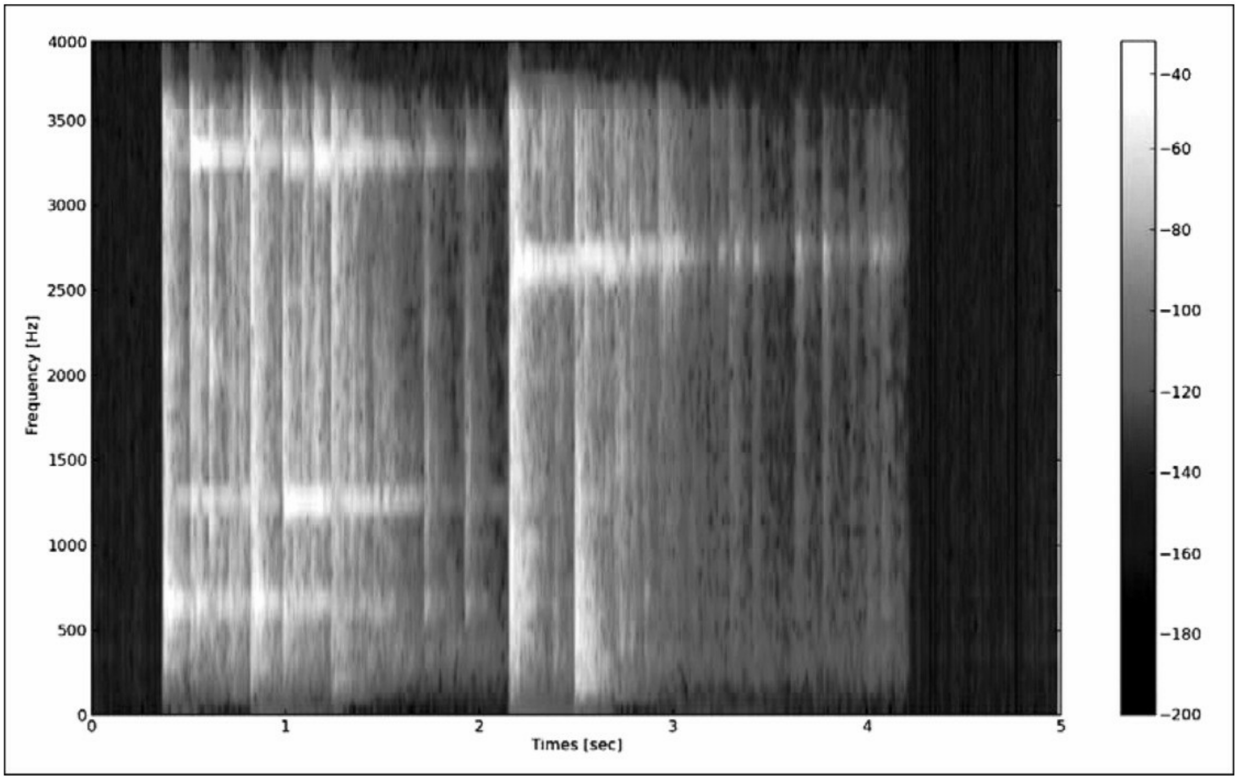


图7-2

NFFT定义了每一个块中用于计算离散傅里叶变换的数据点的数量。当NFFT的值为2的幂次方时计算起来效率最高。窗口可以重叠，重叠（也就是重复）的数据点数量通过参数noverlap指定。

7.3.3 工作原理

首先需要加载一个声音文件，这通过调用 scikits.audiolab.SndFile 方法并传入一个文件名来完成。该方法将实例化一个声音对象，通过该对象可以查询数据以及调用其上的方法。

为了读取频谱图需要的数据，需要从声音对象中读取所需的数据帧。这通过read_frames()完成，该方法接收开始帧和结束帧的参数。把

采样率和想要可视化的时间点（start, end）相乘可以计算出帧数量。

7.3.4 补充说明

如果找不到音频文件（wave），你可以很容易地生成一个。生成方法如下。

```
import numpy
def _get_mask(t, t1, t2, lvl_pos, lvl_neg):
    if t1 >= t2:
        raise ValueError("t1 must be less than t2")
    return numpy.where(numpy.logical_and(t > t1, t < t2), lvl_pos,
lvl_neg)
def generate_signal(t):
    sin1 = numpy.sin(2 * numpy.pi * 100 * t)
    sin2 = 2 * numpy.sin(2 * numpy.pi * 200 * t)
    # add interval of high pitched signal
    sin2 = sin2 * get_mask（t,2,5,1.0,0.0）
    noise = 0.02 * numpy.random.randn(len(t))
    final_signal = sin1 + sin2 + noise
    return final_signal
if __name__ == '__main__':
    step = 0.001
    sampling_freq=1000
    t = numpy.arange(0.0, 20.0, step)
    y = generate_signal(t)
    # we can visualize this now
    # in time
```

```
ax1 = plt.subplot(211)
plt.plot(t, y)
# and in frequency
plt.subplot(212)
plt.specgram(y, NFFT=1024, noverlap=900,
    Fs=sampling_freq, cmap=plt.cm.gist_heat)
plt.show()
```

这将生成如图7-3所示的信号，其中顶部的图形是生成的信号。这里，x轴表示时间，y轴表示信号的幅值。底部的图形是相同的信号在频率域中的呈现。这里，x轴如顶部图一样表示时间（通过选择采样率来匹配时间），y轴表示信号的频率。

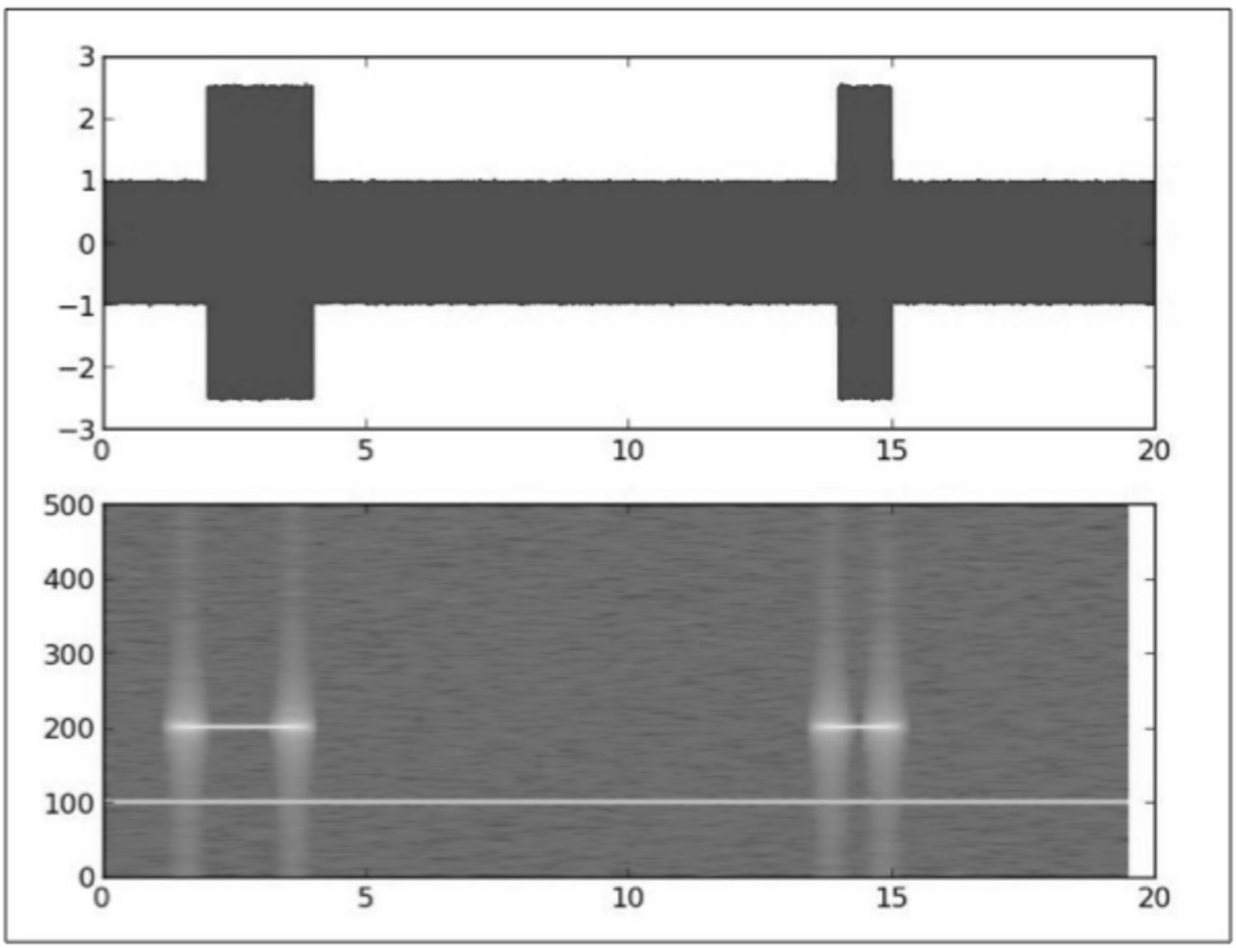


图7-3

7.4 创建火柴杆图

一个二维的火柴杆图（stem plot）把数据显示为沿 x 轴的基线延伸的线条。圆圈（默认值）或者其他标记表示每个杆的结束，其y轴表示了数据值。

本节将讨论如何创建火柴杆图。

不要把火柴杆图和茎叶图（stem and leaf plot）混淆，茎叶图是把最不重要的数值表示为叶，把较高位的值表示为茎的一种数据表现方法，如图7-4所示。

```
steam | leaf
==========================
    0 | 6 7 8
    1 | 0 2 3 4 7 7 7 8 9
    2 | 1 3 4 4 5 7
    3 | 3 1 1 2 6 6 9
    4 | 1 5 5 6 9
    5 | 0
```

图7-4

7.4.1 准备工作

我们想使用一个离散值序列来绘制火柴杆图，普通的线性图表无法

用来展示这种离散的数据。

绘制离散序列为杆，数据值表示为每个杆末端的标记。杆从基线（通常在y=0处）延伸到数据点的值。

7.4.2 操作步骤

我们将使用matplotlib的stem()函数绘制火柴杆图。这个函数可以只使用一系列的y值，x值为生成的一个从0到len(y)-1的简单序列。如果把x和y序列都提供给stem()函数，该函数会把它们都用于两个坐标轴。

我们要为火柴杆图配置下面的一些格式器。

◆ linefmt：这是杆线的线条格式器。

◆ markerfmt：火柴杆线条末端的标记用该参数格式化。

◆ basefmt：规定基线的外观。

◆ label：设置火柴杆图图例的标签。

◆ hold：把所有当前图形放在当前坐标轴上。

◆ bottom：在 y 轴方向设置基线位置，默认值为 0。

参数 hold 被用作图表的一个常见的特性。如果它是打开状态（True），接下来的所有图表都会被添加到当前坐标轴上。否则，每一个图形会创建新的图表和坐标轴。

执行下面的步骤来创建一个火柴杆图。

1.生成随机噪声数据。

2.设置火柴杆参数。

3.绘制火柴杆。

下面是相应的代码。

```
import matplotlib.pyplot as plt
import numpy as np
# time domain in which we sample
```

```
x = np.linspace(0, 20, 50)
# random function to simulate sampled signal
y = np.sin(x + 1) + np.cos(x ** 2)
# here we can setup baseline position
bottom = -0.1
# True -- hold current axes for further plotting
# False -- opposite. clear and use new figure/plot
hold = False
# set label for legend.
label = "delta"
markerline, stemlines, baseline = plt.stem(x, y, bottom=bottom,
    label=label, hold=hold)
# we use setp() here to setup
# multiple properties of lines generated by stem()
plt.setp(markerline, color='red', marker='o')
plt.setp(stemlines, color='blue', linestyle=':')
plt.setp(baseline, color='grey', linewidth=2, linestyle='-')
# draw a legend
plt.legend()
plt.show()
```

以上代码生成的图形如图7-5所示。

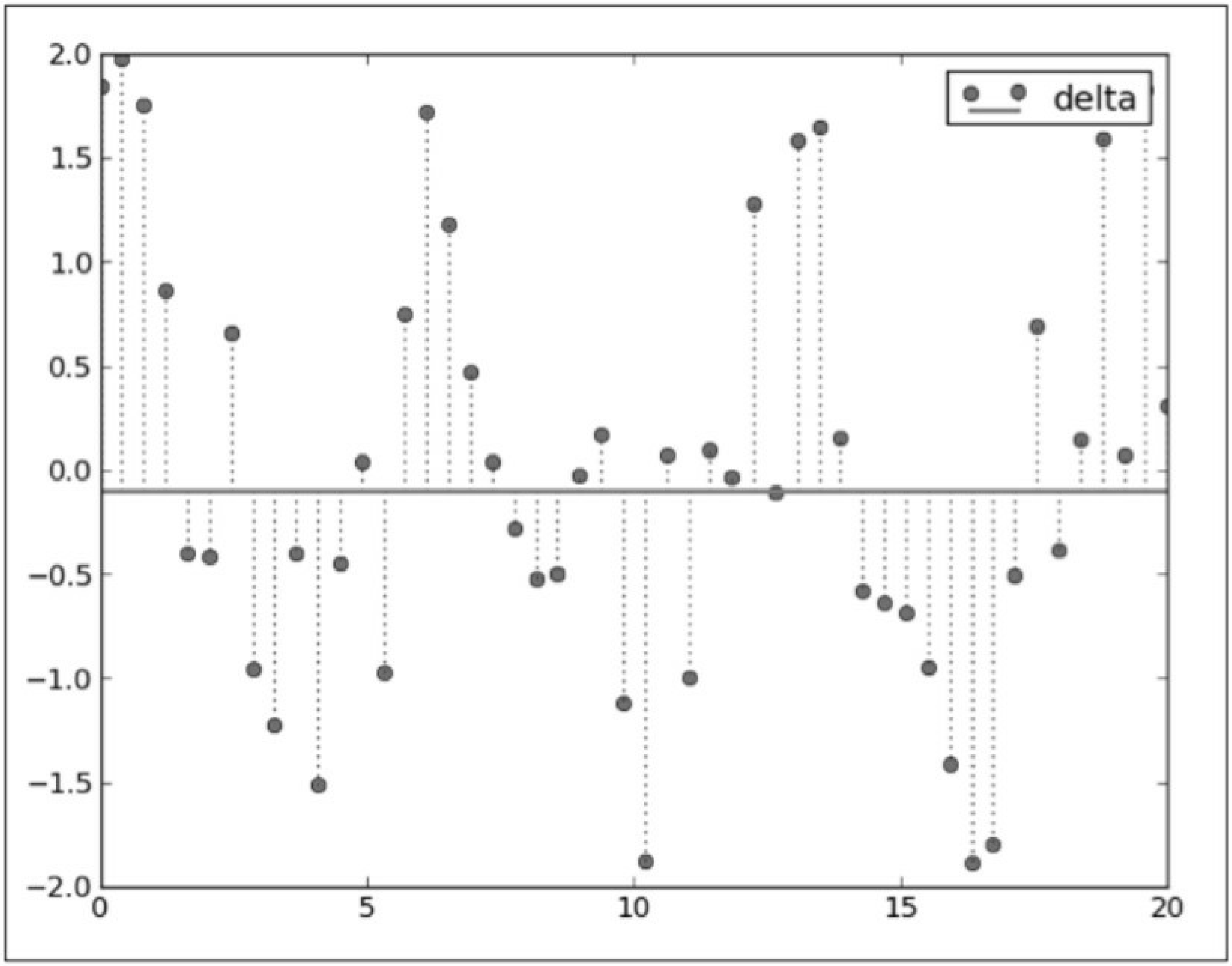


图7-5

7.4.3 工作原理

首先我们需要一些数据。对于本节来说，生成的伪采样信号已经够用了。在真实世界里，任何离散序列数据都可以恰当地用火柴杆图来可视化。我们用 Numpy 的numpy.linspace、numpy.cos和numpy.sin函数生成该信号。

然后，设置火柴杆图的标签和基线的位置，基线位置的默认值为0.0。

如果想要绘制多个火柴杆图，可以设置hold的值为True，这样所有火柴杆图将会被渲染在相同的坐标轴中。

调用matplotlib.stem返回三个对象。第一个是markerline，是一个Line2D的实例，保存了表示火柴杆本身的线条的引用。它仅仅渲染了标记，不包括连接标记的线条。可以通过编辑该Line2D实例的属性让线条可见，操作步骤会在后面解释。最后一个对象 baseline 也是一个 Line2D 实例，保存了表示 stemlines 原点的水平线条的引用。返回的第二个对象是stemlines，当然就是表示茎线的Line2D实例的集合（目前是Python列表）。

通过setp函数把属性应用到这些对象或这些对象集合的所有的线条（Line2D实例）上，我们用这些返回的对象来处理火柴杆图的可视化需求。

你可以尝试一些设置，来理解setp是怎样改变图形风格的。

7.5 绘制矢量场流线图

流线图被用来可视化矢量场的流态。科学和自然学科上的一些例子包括磁场、万有引力和流体运动。

矢量场可以通过为每个点指定一个线条和一个或多个箭头的方式可视化出来。强度可以用线条长度表示，方向由指向特定方向的箭头表示。

通常，力的强度用特定流线的长度显示，但是有时也可以用流线的密度来表示。

7.5.1 准备工作

用matplotlib的matplotlib.pyplot.streamplot函数来可视化矢量场。该函数通过在流场中均匀地填充流线来创建图形。最初该函数是用来可视化风模型或者液体流动的，因此，我们不需要严格的矢量线条，而是需要一个矢量场的统一表现形式。

该函数最重要的参数是（X，Y），是一维 Numpy 数组的等距网格。（U，V）参数匹配的是（X，Y）速率的二维Numpy数组。U和V矩阵在维度上的行数必须等于Y的长度，列的数量必须匹配X的长度。

流线图的线条宽度可以单独控制，如果 linewidth 参数是一个二维数组，将匹配 u和v速率的形状，或者可以是所有线条都可以接受的一个简单的整数。

同样，颜色可以是对于所有流线的一个值，或者像linewidth参数一样形状的一个矩阵。

箭头（FancyArrowPatch 类）用来表示矢量方向，可以通过两个参

数控制它们。arrowsize改变箭头的大小，arrowstyle改变箭头的格式（例如“simple”，“->”…）。

7.5.2 操作步骤

我们从一个简单的例子开始，来了解一下流线图，执行下面的步骤。

1.创建矢量数据。

2.打印中间值。

3.绘制流线图。

4.显示用来可视化矢量的流线的图形。

下面是示例代码。

```
import matplotlib.pyplot as plt
import numpy as np
Y, X = np.mgrid[0:5:100j, 0:5:100j]
U = X
V = Y
from pprint import pprint
print "X"
pprint(X)
print "Y"
pprint(Y)
plt.streamplot(X, Y, U, V)
plt.show()
```

上述代码会输出以下文本信息。

```
X
array([[ 0.        ,  0.05050505,  0.1010101 , ...,  4.8989899 ,
```

```
    4.94949495,   5.        ],
   [ 0.        ,  0.05050505,  0.1010101 , ...,  4.8989899 ,
    4.94949495,   5.        ],
   [ 0.        ,  0.05050505,  0.1010101 , ...,  4.8989899 ,
    4.94949495,   5.        ],
   ...,
   [ 0.        ,  0.05050505,  0.1010101 , ...,  4.8989899 ,
    4.94949495,   5.        ],
   [ 0.        ,  0.05050505,  0.1010101 , ...,  4.8989899 ,
    4.94949495,   5.        ],
   [ 0.        ,  0.05050505,  0.1010101 , ...,  4.8989899 ,
    4.94949495,   5.        ]])
Y
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
    0.        ,  0.        ],
   [ 0.05050505,  0.05050505,  0.05050505, ...,  0.05050505,
    0.05050505,  0.05050505],
   [ 0.1010101 ,  0.1010101 ,  0.1010101 , ...,  0.1010101 ,
    0.1010101 ,  0.1010101 ],
   ...,
   [ 4.8989899 ,  4.8989899 ,  4.8989899 , ...,  4.8989899 ,
    4.8989899 , 4.8989899 ],
   [ 4.94949495, 4.94949495, 4.94949495, ..., 4.94949495,
    4.94949495, 4.94949495],
   [ 5.        ,  5.        ,  5.        , ...,  5.        ,
    5.        ,  5.        ]])
```

生成的流线图图表如图7-6所示。

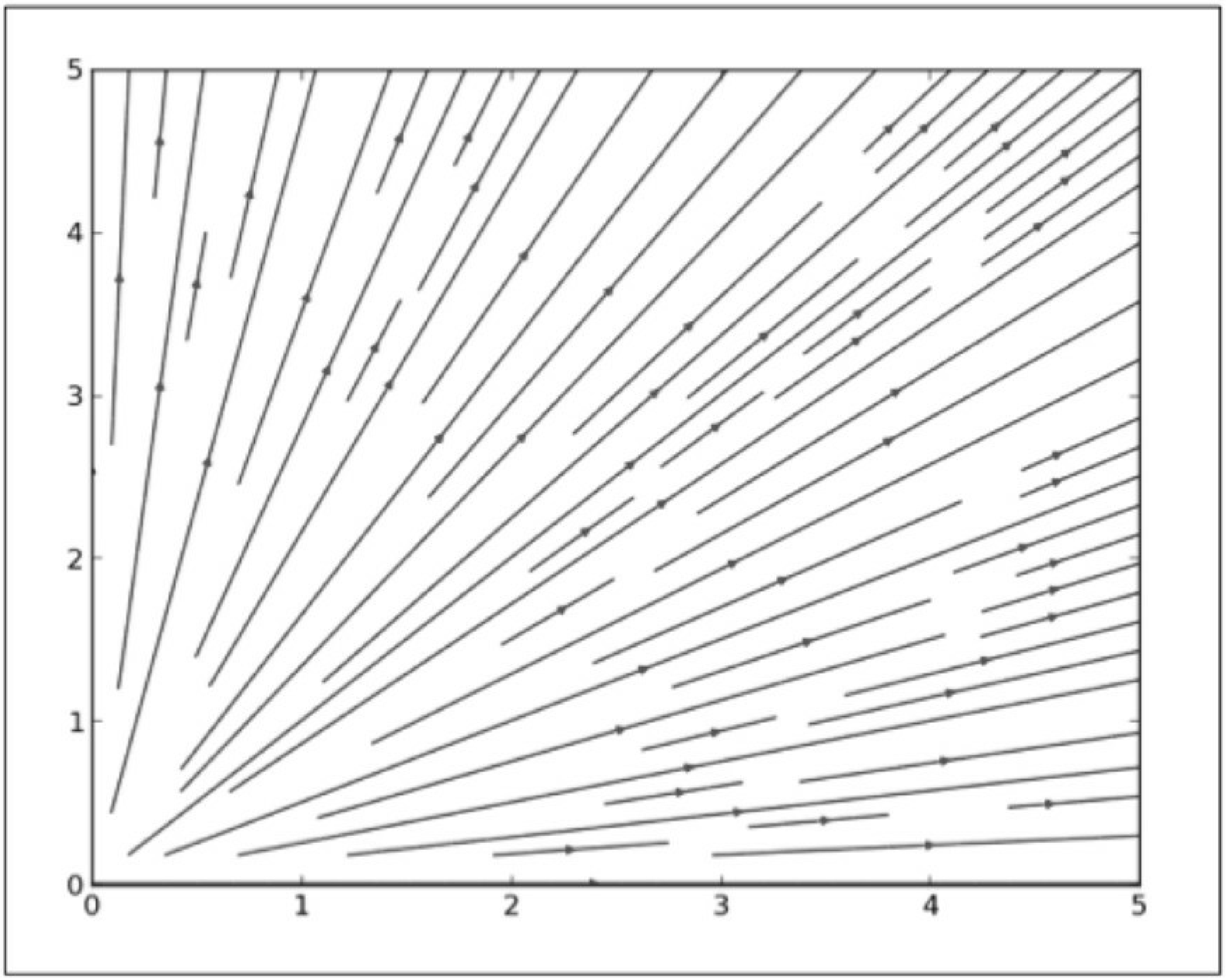


图7-6

7.5.3 工作原理

使用Numpy的mgrid实例，通过检索二维的网状栅格，我们创建了X和Y的矢量场。指定网格的范围作为起点和终点（相应的为-2和2）。第三个索引表示步长。步长表示的是起点和终点之间包含的点的数量。如果想要包含终点值，可以使用一个复数作为步长，其中幅值用于起点和终点之间需要的点数量，终点包含在内。

然后，如上填充的网状栅格被用于计算矢量的速率。这里，为了示例的原因，我们就简单地使用相同的meshgrid属性作为矢量速率。这将生成一个图形，该图形清晰地显示了矢量场的线性依赖和流。

改变一下 U 和 V 的值，体会一下 U 和 V 的值是如何影响流线图的。例如，让 U =np.sin(X)或者 V = sin(Y)。然后，可以尝试改变起点和终点的值。图 7-7 是 U =np.sin(X)的图形。

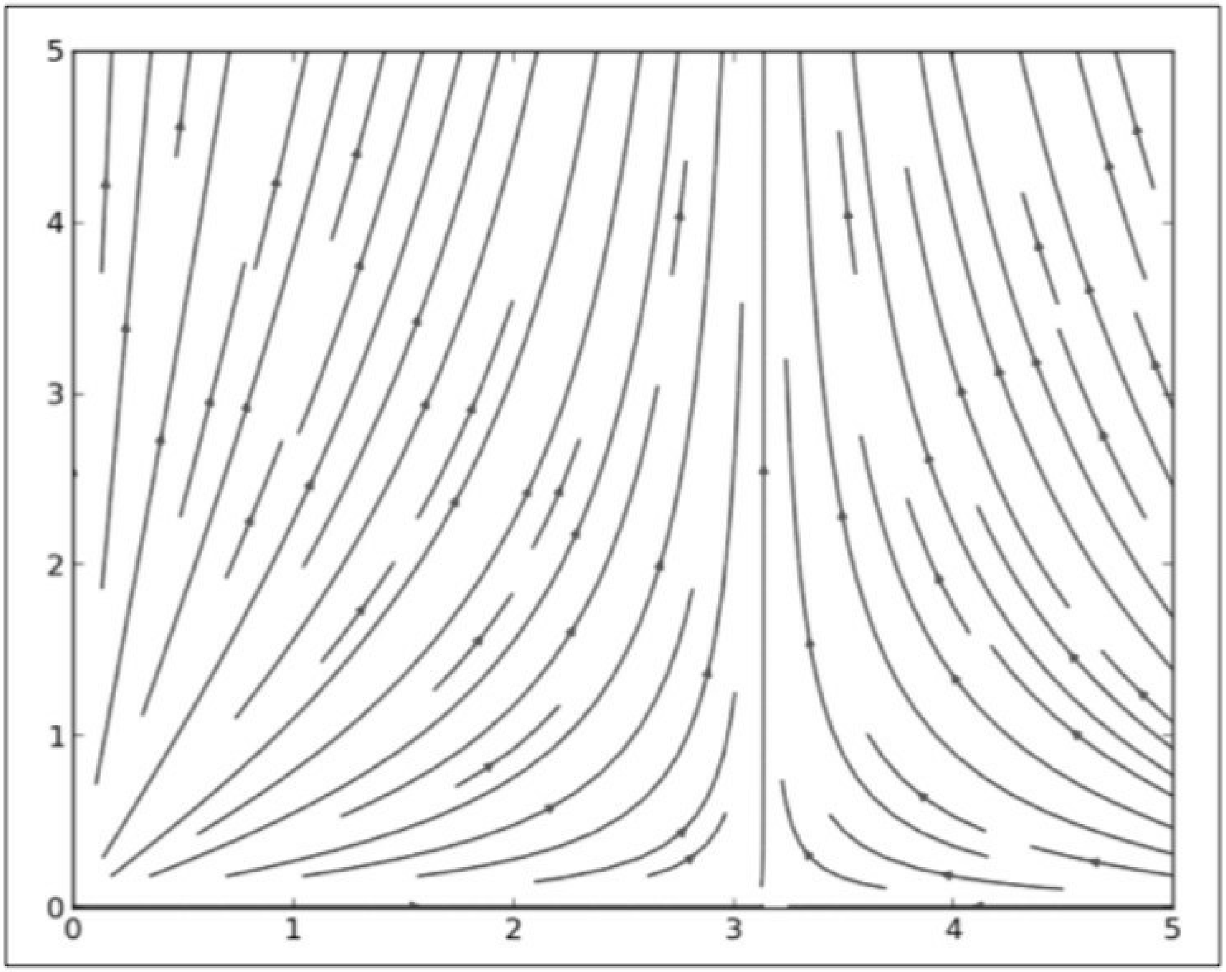


图7-7

要清楚该图表是生成的线条和箭头补片的集合，因此没有办法（至少现在）更新现有的图形，因为线条和箭头对于矢量和场一无所知。将来的实现可能会包含它，但是目前这是matplotlib现有版本的一个公认的局限。

7.5.4 补充说明

当然，这个例子只是给我们一个机会来知道并理解matplotlib的流线

图特性和能力。

当你手头有真正的数据时就会体现其真正的威力。理解了本节内容后，你就能知道你所拥有的工具是什么，这样当给你数据并且了解了它所属的领域时，你就能够选用最适合的工具来完成工作。

7.6 使用颜色表

用颜色来编码数据会极大地影响观察者如何理解可视化图形，因为观察者们会对颜色和颜色要表达的信息做一定的假设。

坦白来讲，如果用颜色向数据添加额外的信息，这终归是件好事。最好也要知道何时以及如何在你的可视化中使用颜色。

7.6.1 准备工作

如果你的数据不是天然用颜色标示的（如地形/地势海拔或者物体的温度），最好不要人为的把它映射到自然色上。我们想要恰当地理解数据，因此选择一种能帮助读者容易地理解数据的颜色。如果展示与开氏温度或摄氏温度无关的财务数据，那么我们不希望读者不断地把数据映射到表示温度的颜色上去。

如果数据没有与红色/绿色有很强的关联时，要尽可能地避免使用这两种颜色。

为了帮助读者选择合适的颜色映射，我们将解释 matplotlib 包中已有的一些颜色表，如果你知道它们是用来做什么的以及怎么找到它们，可以帮助你并节省很多的时间。

颜色表一般可以归为以下几类。

◆ Sequential：这表示同一颜色从低饱和度到高饱和度的两个色调的单色颜色表，例如从白色到天蓝色。对大多数情况来说这是理想的，因为这些颜色清晰地显示了颜色值从低到高的变化。

◆ Diverging：这表示中间值，是颜色的中值（通常是一些明亮的颜色），然后颜色范围在高和低两个方向上变化到两个不同的色调。这

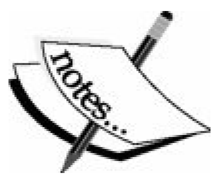
对于有明显中值的数据是理想的。例如，当中值是0时，能清晰地显示负值和正值之间的区别。

◆ Qualitative：对于数据没有固定的顺序，并且想要让不同种类的数据之间能轻易地区分开的情况，可以选用该颜色表。

◆ Cyclic：当数据可以围绕端点值显示的时候，用该颜色表非常方便，例如表示一天的时间、风向或者相位角。

matplotlib自带许多预定义的颜色表，我们可以把它们划分为几类。我们会为何时使用何种颜色表给出一些建议。最基本且常用的颜色表有autumn、bone、cool、copper、flag、gray、hot、hsv、jet、pink、prism、sprint、summer、winter和spectral。

在Yorick科学可视化包中还有其他一些颜色表。这是从GIST包演变而来的，因此在该集合中的所有颜色表名字中都有一个gist_前缀。



Yorick科学可视化包也是一个由C编写的解释型语言，最近不是非常活跃。可以在其官网http://yorick.sourceforge.net/index.php得到更多的信息。

这些颜色表集合包括以下表：gist_earth、gist_heat、gist_ncar、gist_rainbow和gist_stern。

下面介绍基于ColorBrewer（http://colorbrewer.org）的颜色表，可以把它们分为以下几类。

◆ Diverging：中间亮度最高，向两端递减。

◆ Sequential：亮度单调地递减。

◆ Qualitative：不同种类的颜色用来区分不同的数据类别。

另外还有一些可用的颜色表如表7-1所示。

表7-1

颜　色　表	描　　述
brg	这表示一个发散型的蓝—红—绿颜色表
bwr	这表示一个发散型的蓝—白—红颜色表
coolwarm	对于 3D 阴影，色盲和颜色排序非常有用
rainbow	表示一个有发散亮度的紫—蓝—绿—黄—橙—红光谱颜色表
seismic	表示一个发散型蓝—白—红颜色表
terrain	表示地图标记的颜色（蓝、绿、黄、棕和白），最初来自 IGOR Pro 软件

这里展示的大多数颜色表可以通过在颜色表名字后面加上_r 后缀进行反转，例如hot_r是反向循环的hot颜色表。

7.6.2 操作步骤

在 matplotlib 中可以为许多项目设置颜色表。例如，颜色表可以设置在 image, pcolor和 scatter 上。通过在 cmap 函数调用时传入一个参数来完成。参数接受一个colors.Colormap的实例。

也可以使用matplotlib.pyplot.set_cmap为绘制在坐标轴上的最新对象设置cmap。

通过 matplotlib.pyplot.colormaps 可以很容易地得到所有可用的颜色表。打开IPython，输入以下代码。

```
In [1]: import matplotlib.pyplot as plt
In [2]: plt.colormaps()
Out[2]:
  ['Accent',
  'Accent_r',
  'Blues',
  'Blues_r',
  ...
  'winter',
```

```
    'winter_r']
```

注意，我们缩短了上面的输出列表，因为它包含了大约140个元素，会占用好几页。

上述代码将导入pyplot函数接口，允许调用colormaps函数，colormaps函数返回一个所有已注册的颜色表的列表。

最后，我们想向你展示如何创建一个美观的颜色表。在下面的例子中我们需要进行以下操作。

1.打开ColorBrewer网站，得到十六进制格式的diverging颜色表颜色值。

2.生成随机样本x和y，其中y为所有值的累积和（模拟股票价格变动）。

3.在matplotlib的散点图函数上做一些定制化。

4.改变散点标记线条颜色和宽度，使读者更容易理解。

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
# Red Yellow Green divergent colormap
red_yellow_green = ['#d73027', '#f46d43', '#fdae61',
    '#fee08b', '#ffffbf', '#d9ef8b',
    '#a6d96a', '#66bd63', '#1a9850']
sample_size = 1000
fig, ax = plt.subplots(1)
for i in range(9):
    y = np.random.normal(size=sample_size).cumsum()
    x = np.arange(sample_size)
    ax.scatter(x, y, label=str(i), linewidth=0.1,
    edgecolors='grey',
```

```
        facecolor=red_yellow_green[i])
ax.legend()
plt.show()
```

上述代码将渲染出一个漂亮的图表，如图7-8所示。

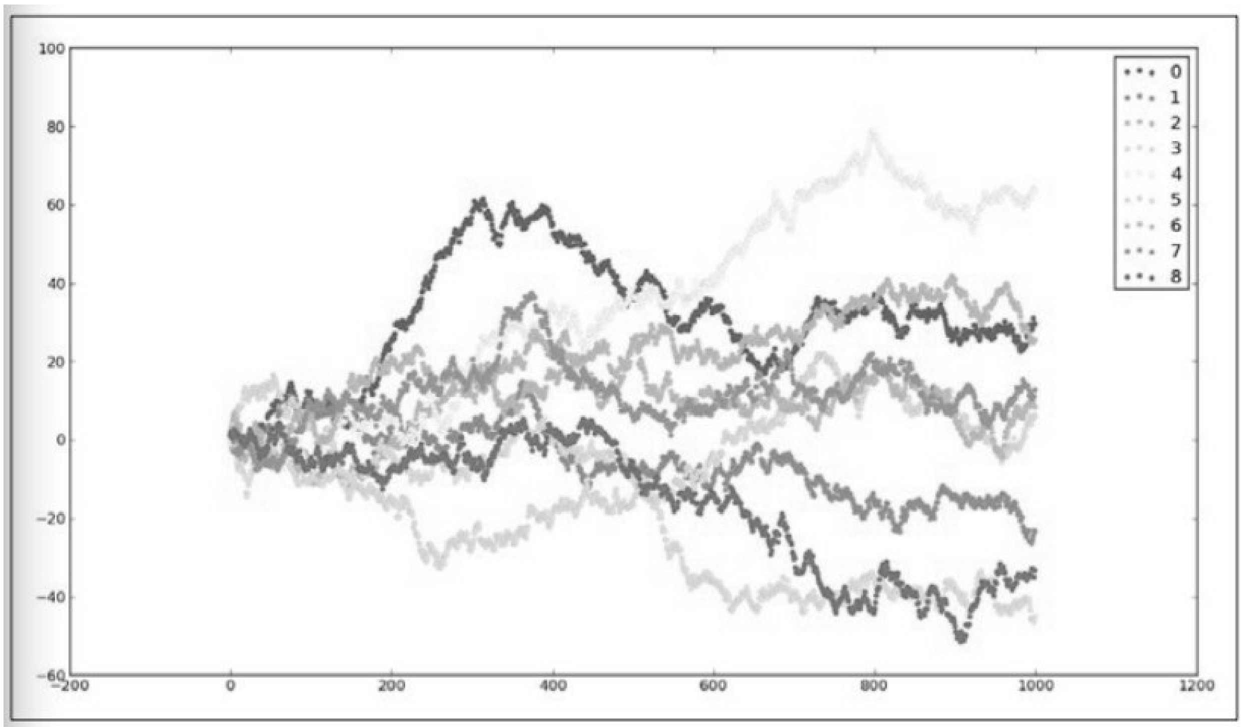
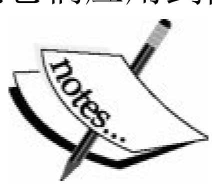


图7-8

7.6.3 工作原理

从ColorBrewer网站找到红—黄—绿diverging颜色表的颜色。然后，在代码中列出这些颜色，并把它们应用到散点图中。



ColorBrewer 是一个由 Cynthia Brewer、Mark Harrower 编写的 Web 工具，宾夕法尼亚州立大学开发了其中的颜色表。这是一个非常好用的工具，可以选择不同范围的颜色表并把它们应用在地图上，观察它们的

不同。这样，可以快速地了解它们显示在一个图表上的样子。这个独特的地图地址是 http://colorbrewer2.org/index.php?type=diverging&scheme=RdYlGn&n=9。

有时候，我们不得不在 matplotlib.rcParams 上做一些定制化，这是在创建一个图表或者任何坐标轴之前要做的第一件事情。

例如，为了为大多数 matplotlib 函数设置默认的颜色表，需要改变配置参数 matplotlib. rcParams['axes.cycle_color']。

7.6.4 补充说明

通过 matplotlib.pyplot.register_cmap ，可以将一个新的颜色表注册到matplotlib，这样就可以通过get_cmap函数找到它。我们可以通过两种不同的方式使用它，两种签名形式如下。

- register_cmap(name='swirly', cmap=swirly_cmap)
- register_cmap(name='choppy', data=choppydata, lut=128)

第一种签名指定一个颜色表作为colors.Colormap的实例，并通过name参数注册。参数name可以忽略，在这种情况下，它将继承cmap实例提供的name属性。

对于第二种签名，我们向线性分隔的颜色表构造函数传入三个参数，随后把该颜色表注册到matplotlib。

我们可以通过把name参数传入matplotlib.pyplot.get_cmap函数来得到相应的colors.Colormap实例。

下面的代码向展示如何使用 matplotlib.colors.LinearSegmentedColormap创建你自己的颜色表：

```
from pylab import *
cdict = {'red': ((0.0, 0.0, 0.0),
   (0.5, 1.0, 0.7),
```

```
    (1.0, 1.0, 1.0)),
    'green': ((0.0, 0.0, 0.0),
      (0.5, 1.0, 0.0),
      (1.0, 1.0, 1.0)),
    'blue': ((0.0, 0.0, 0.0),
      (0.5, 1.0, 0.0),
      (1.0, 0.5, 1.0))}
my_cmap = matplotlib.colors.LinearSegmentedColormap('my_
colormap',cdict,256)
pcolor(rand(10,10),cmap=my_cmap)
colorbar()
```

执行该方法很简单，实际上难的部分是给出信息丰富的颜色组合，这种颜色组合不会从我们想要可视化的数据中丢掉任何信息，同时让读者看起来赏心悦目。

对于基本颜色列表（在之前的表中列出的颜色表），可以用pylab快捷方式来设置颜色表。例如：

```
imshow(X)
hot()
```

这将设置图像 X 的颜色表为 cmap = 'hot'。

7.7 使用散点图和直方图

我们经常会遇到散点图，因为它们是可视化两个变量之间关系时最常用的图表。如果想快速地查看两个变量的数据，并看看它们之间是否有关系（也就是相关性），我们可以快速绘制一个散点图。对于一个散点图，必须有一个变量可以被改变，比如说，实验者有系统地改变这个变量，这样就可以观察到它对另一个变量可能产生的影响。

这就是我们在本节中要学习如何理解散点图的原因。

7.7.1 准备工作

例如，我们想看两个事件是怎么相互影响的，或者它们是否真的相互影响。这种可视化在大数据集合上尤其有用，因为当只有数据时，我们没有办法通过查看原生格式的数据得到任何结论。

如果数值之间存在相关性，这种相关性可以是正相关或负相关。正相关指在增大 X 的值时，Y的值也会增加。负相关时增加X的值，Y的值会减小。在理想情况下，正相关是一条从坐标轴的左下角到右上角的线段。理想的负相关是一条从坐标轴的左上角到右下角的线段。

两个数据点之间理想的正相关是值为1，理想的负相关是值为-1。所有在此区间内的值表示两个值之间存在较弱的相关性。通常，从两个变量的真正关联的角度看，在-0.5～0.5之间的值被认为是没有价值的。

一个正相关的例子是，放到慈善罐中的钱的总数直接与看到罐子的人数呈正相关性。一个负相关的例子是，从地点B到地点A所需要的时间，取决于地点A与地点B之间的距离。距离越大，完成这段旅行所花费的时间也越多。

我们这里展示的正相关的例子并不完美，因为每次访问时，不同的人放的钱的数量可能不同。但是一般来讲，我们可以假定看到罐子的人数量越多，罐子里的钱就越多。

但是要记住，即使散点图显示了两个变量间存在相关性，但是它可能不是一个直接相关。可能有第三个变量影响所绘制的两个变量，因此相关性就仅仅是绘制的变量与那第三个变量相关。最后，相关性也许仅仅是看上去明显，但是在其背后并不存在真正的关系。

7.7.2 操作步骤

通过下面的示例代码，我们将展示散点图如何解释变量间的关联。

我们使用的数据是从 Google Trends 门户网站获得，在那里可以下载到包含给定参数的相关搜索量的归一化值的CSV文件。

将数据存储在 ch07_search_data.py Python模块中，这样就可以在接下来的代码中导入它。内容如下。

```
# ch07_search_data
# daily search trend for keyword 'flowers' for a year
DATA = [
    1.04, 1.04, 1.16, 1.22, 1.46, 2.34, 1.16, 1.12, 1.24, 1.30, 1.44,
    1.22, 1.26,
    1.34, 1.26, 1.40, 1.52, 2.56, 1.36, 1.30, 1.20, 1.12, 1.12, 1.12,
    1.06, 1.06,
    1.00, 1.02, 1.04, 1.02, 1.06, 1.02, 1.04, 0.98, 0.98, 0.98, 1.00,
    1.02, 1.02,
    1.00, 1.02, 0.96, 0.94, 0.94, 0.94, 0.96, 0.86, 0.92, 0.98, 1.08,
    1.04, 0.74,
    0.98, 1.02, 1.02, 1.12, 1.34, 2.02, 1.68, 1.12, 1.38, 1.14, 1.16,
```

1.22, 1.10,

1.14, 1.16, 1.28, 1.44, 2.58, 1.30, 1.20, 1.16, 1.06, 1.06, 1.08, 1.00, 1.00,

0.92, 1.00, 1.02, 1.00, 1.06, 1.10, 1.14, 1.08, 1.00, 1.04, 1.10, 1.06, 1.06,

1.06, 1.02, 1.04, 0.96, 0.96, 0.96, 0.92, 0.84, 0.88, 0.90, 1.00, 1.08, 0.80,

0.90, 0.98, 1.00, 1.10, 1.24, 1.66, 1.94, 1.02, 1.06, 1.08, 1.10, 1.30, 1.10,

1.12, 1.20, 1.16, 1.26, 1.42, 2.18, 1.26, 1.06, 1.00, 1.04, 1.00, 0.98, 0.94,

0.88, 0.98, 0.96, 0.92, 0.94, 0.96, 0.96, 0.94, 0.90, 0.92, 0.96, 0.96, 0.96,

0.98, 0.90, 0.90, 0.88, 0.88, 0.88, 0.90, 0.78, 0.84, 0.86, 0.92, 1.00, 0.68,

0.82, 0.90, 0.88, 0.98, 1.08, 1.36, 2.04, 0.98, 0.96, 1.02, 1.20, 0.98, 1.00,

1.08, 0.98, 1.02, 1.14, 1.28, 2.04, 1.16, 1.04, 0.96, 0.98, 0.92, 0.86, 0.88,

0.82, 0.92, 0.90, 0.86, 0.84, 0.86, 0.90, 0.84, 0.82, 0.82, 0.86, 0.86, 0.84,

0.84, 0.82, 0.80, 0.78, 0.78, 0.76, 0.74, 0.68, 0.74, 0.80, 0.80, 0.90, 0.60,

0.72, 0.80, 0.82, 0.86, 0.94, 1.24, 1.92, 0.92, 1.12, 0.90, 0.90, 0.94, 0.90,

0.90, 0.94, 0.98, 1.08, 1.24, 2.04, 1.04, 0.94, 0.86, 0.86, 0.86, 0.82, 0.84,

0.76, 0.80, 0.80, 0.80, 0.78, 0.80, 0.82, 0.76, 0.76, 0.76, 0.76,
0.78, 0.78,
0.76, 0.76, 0.72, 0.74, 0.70, 0.68, 0.72, 0.70, 0.64, 0.70, 0.72,
0.74, 0.64,
0.62, 0.74, 0.80, 0.82, 0.88, 1.02, 1.66, 0.94, 0.94, 0.96, 1.00,
1.16, 1.02,
1.04, 1.06, 1.02, 1.10, 1.22, 1.94, 1.18, 1.12, 1.06, 1.06, 1.04,
1.02, 0.94,
0.94, 0.98, 0.96, 0.96, 0.98, 1.00, 0.96, 0.92, 0.90, 0.86, 0.82,
0.90, 0.84,
0.84, 0.82, 0.80, 0.80, 0.76, 0.80, 0.82, 0.80, 0.72, 0.72, 0.76,
0.80, 0.76,
0.70, 0.74, 0.82, 0.84, 0.88, 0.98, 1.44, 0.96, 0.88, 0.92, 1.08,
0.90, 0.92,
0.96, 0.94, 1.04, 1.08, 1.14, 1.66, 1.08, 0.96, 0.90, 0.86, 0.84,
0.86, 0.82,
0.84, 0.82, 0.84, 0.84, 0.84, 0.84, 0.82, 0.86, 0.82, 0.82, 0.86,
0.90, 0.84,
0.82, 0.78, 0.80, 0.78, 0.74, 0.78, 0.76, 0.76, 0.70, 0.72, 0.76,
0.72, 0.70,
0.64]

我们需要执行下面的步骤。

1.使用一个干净的数据集合，该集合是对关键字 flowers 在 Google Trend 上一年的搜索量，把该数据集合导入到变量d中。

2.使用一个相同长度（365 个数据点）的随机正态分布作为 Google Trend 数据集合，这个集合为d1。

3.创建包含4个子区的图表。

4.在第一个子区中，绘制d和d1的散点图。

5.在第二个子区中，绘制d1和d1的散点图。

6.在第三个子区中，绘制d1和反序d1的散点图。

7.在第四个子区中，绘制d1和由d1和d组合而成的数据集合的散点图。

下面的代码演示了本节中前面部分解释的关系：

```
import matplotlib.pyplot as plt
import numpy as np
# import the data
from ch07_search_data import DATA
d = DATA
# Now let's generate random data for the same period
d1 = np.random.random(365)
assert len(d) == len(d1)
fig = plt.figure()
ax1 = fig.add_subplot(221)
ax1.scatter(d, d1, alpha=0.5)
ax1.set_title('No correlation')
ax1.grid(True)
ax2 = fig.add_subplot(222)
ax2.scatter(d1, d1, alpha=0.5)
ax2.set_title('Ideal positive correlation')
ax2.grid(True)
ax3 = fig.add_subplot(223)
ax3.scatter(d1, d1*-1, alpha=0.5)
ax3.set_title('Ideal negative correlation')
ax3.grid(True)
```

```
ax4 = fig.add_subplot(224)
ax4.scatter(d1, d1+d, alpha=0.5)
ax4.set_title('Non ideal positive correlation')
ax4.grid(True)
plt.tight_layout()
plt.show()
```

当执行上面代码时，得到如图7-9的输出。

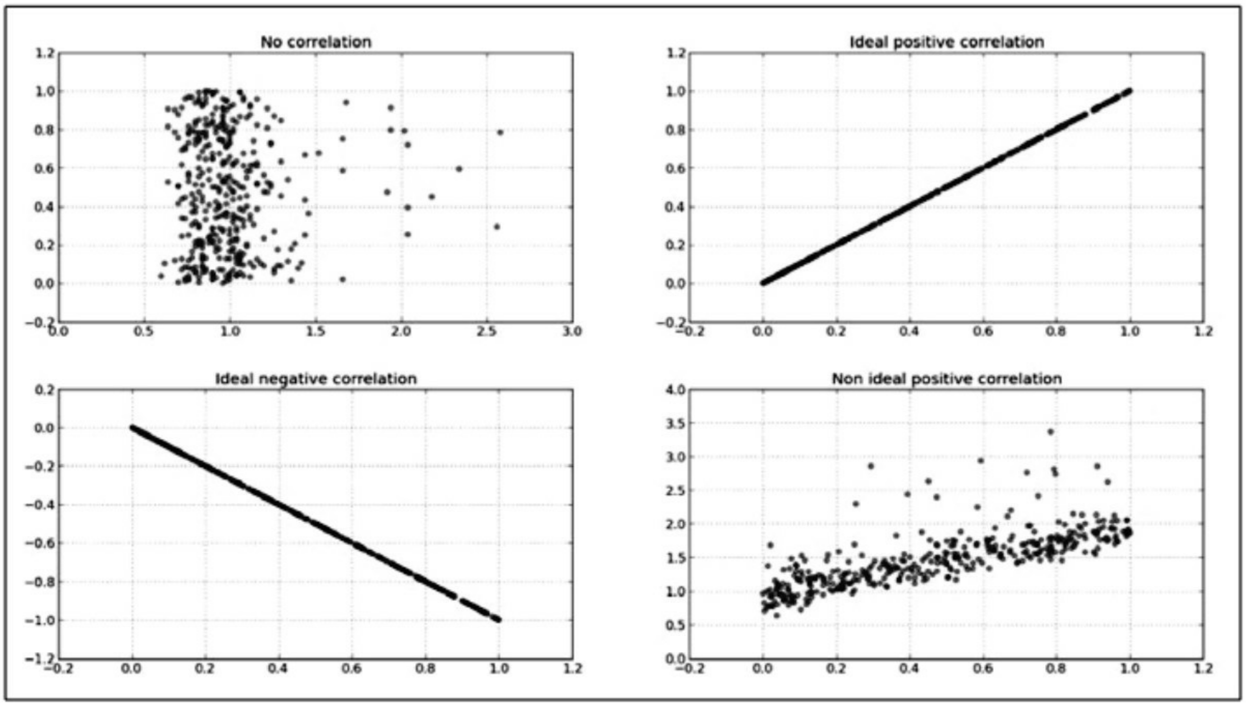


图7-9

7.7.3 工作原理

在上面的输出中，我们清楚地看到在不同的数据集合之间是否存在相关性。其中，第二幅（右上）图显示了一个数据集合d1和d1自身（显然地）之间理想的正相关。第四幅（右下）图表明数据集合间存在一个正相关，虽然不是理想正相关。我们用d1和d（随机的）构建的这个数据集合来模拟两个相似的信号（事件）。在这两幅图中，第二个使用 d

和 d1 绘制的子区图形中有一定的随机性（或者噪声），但还是可以和原始（d）信号进行比较。

7.7.4 补充说明

我们也可以给散点图添加直方图，通过这种方式能了解更多所绘制的数据的信息。我们可以添加水平直方图和垂直直方图来显示在x轴和y轴上数据点的频率。通过这种方法，可以同时看到整个数据集合的汇总信息（直方图）和每一个数据点（散点图）。

下面是一个生成散点—直方图组合的代码示例，使用了在本节中提到的两个相同的数据集合。代码的重点是scatterhist()函数，我们可以给它传入不同的数据集合，它使用所提供的数据集合对一些变量（直方图中bin的数量、坐标轴的范围等）进行设置。

我们从通常的导入开始，代码如下。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

下面代码定义了生成散点直方图的函数，给函数一个（x，y）数据集合和一个可选的figsize参数。

```
def scatterhist(x, y, figsize=(8,8)):
    """
    Create simple scatter & histograms of data x, y inside given plot
    @param figsize: Figure size to create figure
    @type figsize: Tuple of two floats representing size in inches
    @param x: X axis data set
    @type x: np.array
    @param y: Y axis data set
```

```
    @type y: np.array
    """
_, scatter_axes = plt.subplots(figsize=figsize)
    # the scatter plot:
    scatter_axes.scatter(x, y, alpha=0.5)
    scatter_axes.set_aspect(1.)
    divider = make_axes_locatable(scatter_axes)
    axes_hist_x = divider.append_axes(position="top", sharex=scatter_
axes, size=1, pad=0.1)
    axes_hist_y = divider.append_axes(position="right",
sharey=scatter_axes,
        size=1, pad=0.1)
    # compute bins accordingly
    binwidth = 0.25
    # global max value in both data sets
    xymax = np.max([np.max(np.fabs(x)), np.max(np.fabs(y))])
    # number of bins
    bincap = int(xymax / binwidth) * binwidth
    bins = np.arange(-bincap, bincap, binwidth)
    nx, binsx, _ = axes_hist_x.hist(x, bins=bins,
histtype='stepfilled',
        orientation='vertical')
    ny, binsy, _ = axes_hist_y.hist(y, bins=bins,
histtype='stepfilled',
        orientation='horizontal')
    tickstep = 50
    ticksmax = np.max([np.max(nx), np.max(ny)])
```

```
    xyticks = np.arange(0, ticksmax + tickstep, tickstep)
    # hide x and y ticklabels on histograms
    for tl in axes_hist_x.get_xticklabels():
        tl.set_visible(False)
    axes_hist_x.set_yticks(xyticks)
    for tl in axes_hist_y.get_yticklabels():
        tl.set_visible(False)
    axes_hist_y.set_xticks(xyticks)
    plt.show()
```

现在，加载数据并调用函数来生成并渲染出想要的图表。

```
if __name__ == '__main__': # import the data
    from ch07_search_data import DATA as d
    # Now let's generate random data for the same period
    d1 = np.random.random(365)
    assert len(d) == len(d1)
    # try with the random data
    # d = np.random.randn(1000)
    # d1 = np.random.randn(1000)
    scatterhist(d, d1)
```

上述代码将生成如图7-10所示的图表。

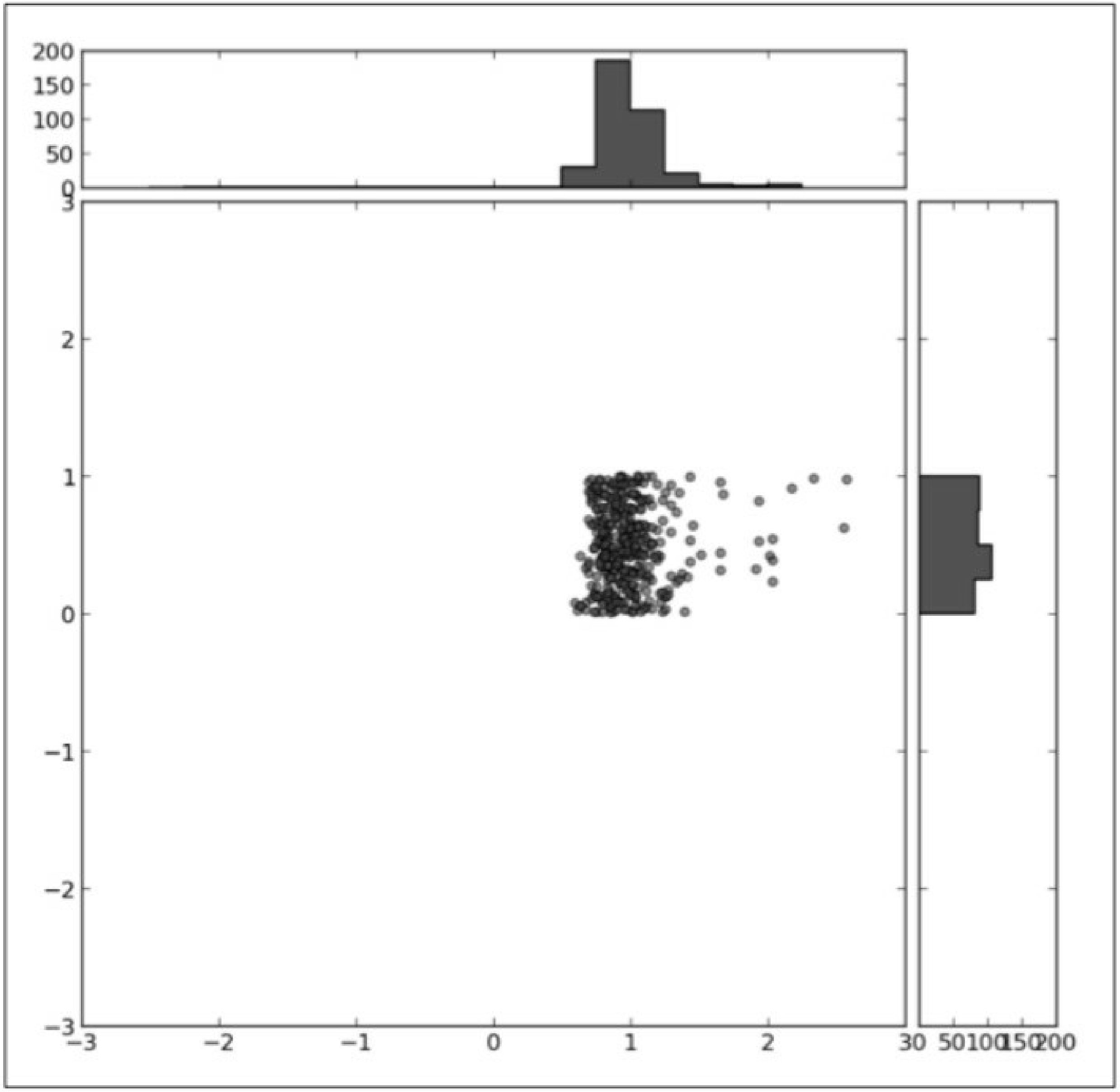


图7-10

7.8 绘制两个变量间的互相关图形

如果有从两个不同的观察结果得到的两个不同数据集合，我们想知道这两个事件集合是否是相关的。我们想把它们交叉关联来看其是否以某种方式匹配。我们在一个较大的数据样本中寻找一个较小数据样本的模式。这个模式不必是明显或者细微的模式。

7.8.1 准备工作

我们可以使用 pyplot lab 中 matplotlib 的 matplotlib.pyplot.xcorr 函数。这个函数可以绘制两个数据集合之间的相互关系，通过这种方式可以看出在绘制的值之间是否存在某个显著的模式。这里假设传入的x和y参数的长度相同。

如果传入的normed参数为True，可以通过0th（也就是说，当没有时间延迟或者时差时）延迟的互关联对数据进行归一化。

在内部，由Numpy的numpy.correlate函数来完成相关性计算。

通过参数usevlines（置为True），我们告诉matplotlib用vlines()而不是plot()绘制相关图形的线条。二者的主要的区别是，如果使用plot()，可以使用标准的Line2D属性设置线条风格，该属性通过**kwargs参数传入matplotlib.pyplot.xcorr函数。

7.8.2 操作步骤

在下面的例子中，我们需要执行以下步骤。

1.导入matplotlib.pyplot模块。

2.导入numpy包。

3.使用一个干净的数据集合，该集合是Google中对关键字flowers一年的搜索量趋势。

4.绘制数据集合（真实的和仿造的）和互相关图表。

5.为了标签和刻度有一个较好的显示效果使用紧凑布局。

6.为了能更容易地理解图表添加恰当的标签和网格。

下面代码将会执行以上提到的步骤。

```
import matplotlib.pyplot as plt
import numpy as np
# import the data
from ch07_search_data import DATA as d
total = sum(d)
av = total / len(d)
z = [i - av for i in d]
# Now let's generate random data for the same period
d1 = np.random.random(365)
assert len(d) == len(d1)
total1 = sum(d1)
av1 = total1 / len(d1)
z1 = [i - av1 for i in d1]
fig = plt.figure()
# Search trend volume
ax1 = fig.add_subplot(311)
ax1.plot(d)
ax1.set_xlabel('Google Trends data for "flowers"')
# Random: "search trend volume"
ax2 = fig.add_subplot(312)
ax2.plot(d1)
```

```
ax2.set_xlabel('Random data')
# Is there a pattern in search trend for this keyword?
ax3 = fig.add_subplot(313)
ax3.set_xlabel('Cross correlation of random data')
ax3.xcorr(z, z1, usevlines=True, maxlags=None, normed=True, lw=2)
ax3.grid(True)
plt.ylim(-1,1)
plt.tight_layout()
plt.show()
```

以上代码将生成如图7-11所示的图表。

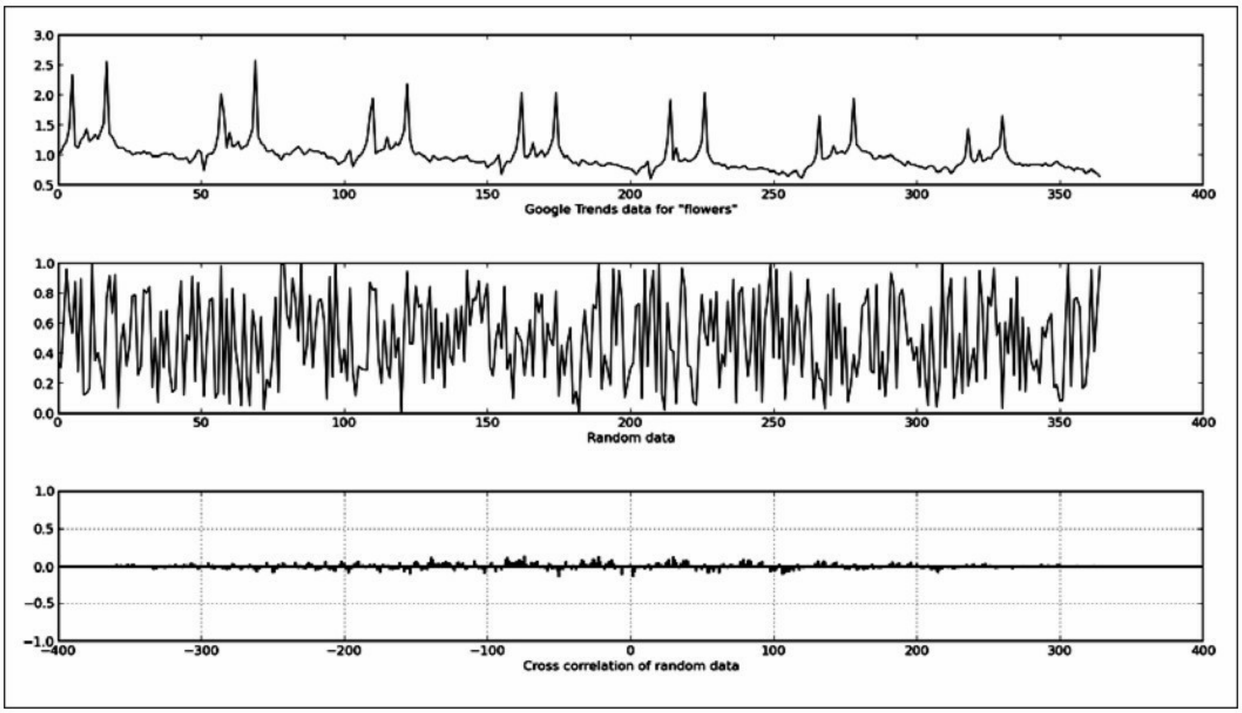


图7-11

7.8.3 工作原理

我们使用了一个有可识别模式（请参考上面的图表，在数据集合上两个峰值以相似的方式重复）的真实数据集合。另一个数据集合仅仅是

一些随机正态分布的数据，该数据和从公共服务 Google Trends 上拿到的真实数据有着相同的长度。

我们把这两个数据集合绘制在输出图表的上半部来对其进行可视化。

使用matplotlib的xcorr函数，转而调用NumPy的correlate()函数，计算出互相关并把其绘制在图表的下半部。

NumPy中的互相关性计算返回一个相关系数数组，该数组表示了两个数据集合（或者如果用在信号处理领域，通常指信号）的相似度。

互相关图表，或者叫相关图，通过相关值的高度（出现在某个时间延迟的竖线）表现，告诉我们这两个信号是不相关的。我们可以看到有不止一个竖线（在时间延迟n上的相关系数）在0.5之上。

举个例子，如果两个数据集合在100s的时间延迟（也就是通过两种不同的传感器观察到的相同对象在相隔100s的两个时间点间的变化）上有相关性，则将在上图输出中x=100的位置上看到一个竖线（表示相关系数）。

7.9 自相关的重要性

自相关表示在一个给定的时间序列或一个连续的时间间隔上其自身的延迟（也就是时间上的延迟）版本之间的相似度。它发生在一个时间序列研究中，指在一个给定的时间周期内的错误在未来的时间周期上继续存在。例如，假如我们在预测股票红利的增长，某一年的过高估计往往会导致对接下来年份的过高估计。

时间序列分析数据引出了许多不同的科学应用和财务流程，一些例子包括生成的财务绩效报表、一段时间的价格、波动性计算等。

如果我们在分析未知数据，自相关可以帮助我们检测数据是否是随机的。对此我们可以使用相关图。它可以提供如下问题的答案：数据是随机的吗？这个时间序列数据时一个白噪声信号吗？它是正弦曲线形的吗？它是自回归的吗？这个时间序列数据的模型是什么？

7.9.1 准备工作

我们将使用 matplotlib 来比较两组数据。一组是某个关键字一年（365 天）的 Google每日趋势的搜索量。另一组是正态分布的365个随机测量值（生成的随机数据）。

我们将分析两个数据集合的自相关性，并比较相关图是如何可视化数据中的模式的。

7.9.2 操作步骤

本小节将执行以下步骤。

1.导入matplotlib.pyplot模块。

2.导入numpy包。

3.使用一个干净的Google一年搜索量的数据集合。

4.绘制数据和其自相关图表。

5.用NumPy生成一个相同长度的随机数据集合。

6.在相同图表上绘制随机数据集合和其自相关图表。

7.添加合适的标签和网格帮助我们理解图表。

下面是代码部分。

```
import matplotlib.pyplot as plt
import numpy as np
# import the data
from ch07_search_data import DATA as d
total = sum(d)
av = total / len(d)
z = [i - av for i in d]
fig = plt.figure()
# plt.title('Comparing autocorrelations')
# Search trend volume
ax1 = fig.add_subplot(221)
ax1.plot(d)
ax1.set_xlabel('Google Trends data for "flowers"')
# Is there a pattern in search trend for this keyword?
ax2 = fig.add_subplot(222)
ax2.acorr(z, usevlines=True, maxlags=None, normed=True, lw=2)
ax2.grid(True)
ax2.set_xlabel('Autocorrelation')
# Now let's generate random data for the same period
d1 = np.random.random(365)
```

```
assert len(d) == len(d1)
total = sum(d1)
av = total / len(d1)
z = [i - av for i in d1]
# Random: "search trend volume"
ax3 = fig.add_subplot(223)
ax3.plot(d1)
ax3.set_xlabel('Random data')
# Is there a pattern in search trend for this keyword?
ax4 = fig.add_subplot(224)
ax4.set_xlabel('Autocorrelation of random data')
ax4.acorr( z, usevlines=True, maxlags=None, normed=True, lw=2)
ax4.grid(True)
plt.show()
```

上述代码将生成如图7-12所示的图表。

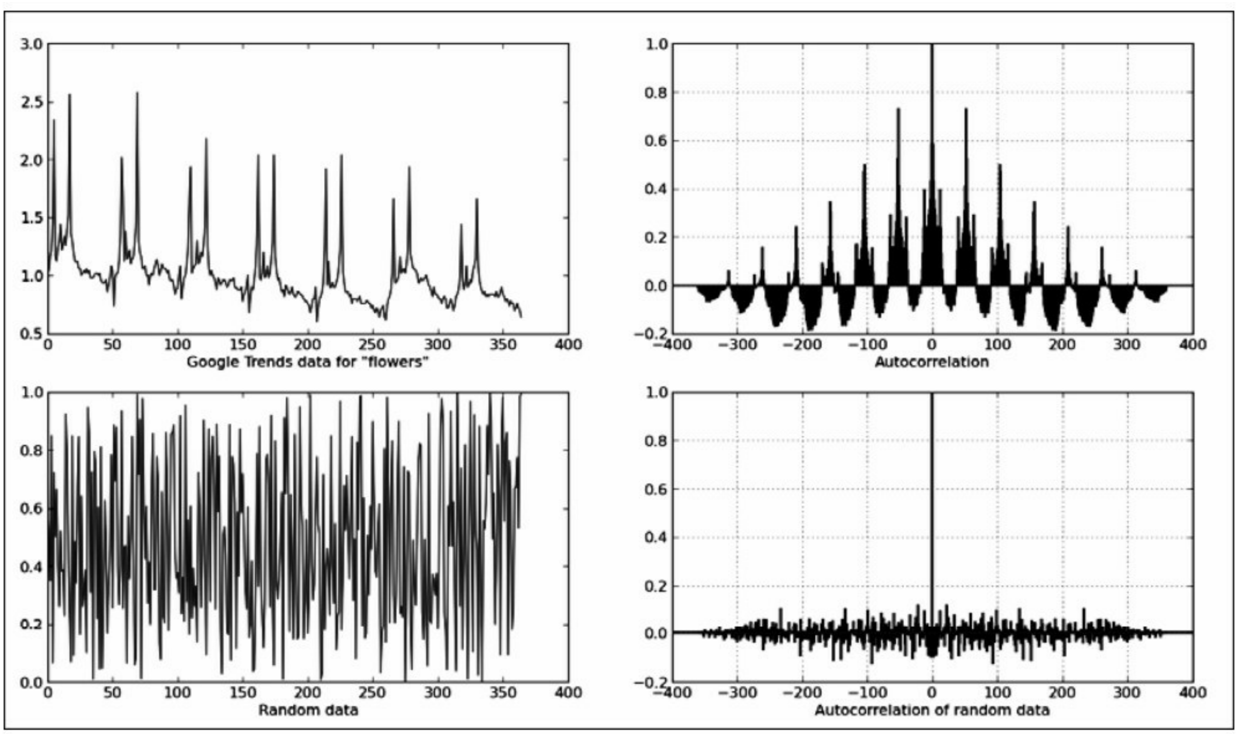


图7-12

7.9.3 工作原理

通过观察左手边的图表，我们能很容易地识别出搜索量数据的模式；左下方的图表指正态分布的随机数据，其模式不是很明显，但仍然是可能存在的。

在随机数据上计算自相关性和绘制自相关图表，可以看到在0处有一个很高的相关性，这是我们所期望的，数据在没有任何时间延迟的时候和自身是相关的。但在无时间延迟之前和之后，信号几乎为 0。因此我们可以安全地推断信号在初始时间和任何观察的时间延迟上没有相关性。

再看一下真实的数据——Google搜索量趋势，我们可以看到在0s时间延迟上有相同的表现，我们也可以预料对于任何自相关信号都会有相同的表现。但是我们看到在0s时间延迟之后的大约30、60和110天存在很强的信号。这表明在Google搜索引擎上这个特殊的搜索术语以及人们搜索它的方式间存在一个模式。

我们把这个为什么这里会存在一个很大的不同的解释工作留给读者。请记住相关和因果关系是两个非常不同的概念。

7.9.4 补充说明

自相关通常应用在当我们想要识别未知数据的模式的时候。当我们想把数据放到一个模型中时，有时候识别我们展示的数据集合的合适模型的第一步是数据是如何与自身相关的。这会需要 Python 以外的知识，它需要数学建模和各种统计测试（Ljung-Box 测试、Box-Pierce测试等）的知识，这些知识能帮助我们解答可能遇到的任何问题。

第8章 更多的matplotlib知识

本章中包含以下内容。

◆ 绘制风杆（barbs）

◆ 绘制箱线图

◆ 绘制甘特图

◆ 绘制误差条

◆ 使用文本和字体属性

◆ 用 LaTeX 渲染文本

◆ 理解 pyplot 和 OO API 的不同

8.1 简介

本章将学习一些matplotlib包中不常使用的特性。其中的一些例子超出了matplotlib最初的目标，但是它们向我们展示了如何做一些创造性的工作，并证明了matplotlib是一个功能全面且一般化的工具。

8.2 绘制风杆（barbs）

风杆是风速和风向的一种表现形式，主要由气象学家使用。理论上讲，它们可以被用来可视化任何类型的二维向量。它们和箭头类似，但不同的是通过箭头的长度表示向量的大小，而风杆通过把直线或者三角形作为大小增量提供了更多关于向量大小的信息。

下面解释风杆是什么、如何理解风杆，以及如何用Python和matplotlib把它们可视化出来。如图8-1所示是一组典型的风杆：

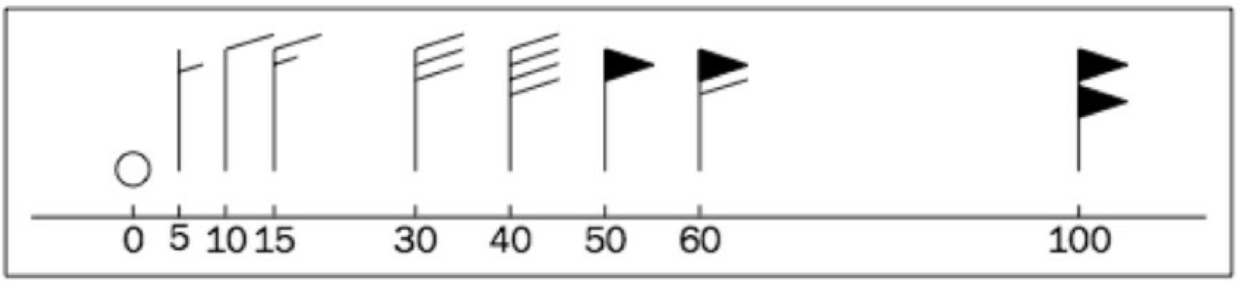


图8-1

在上图中的三角形，或者称为旗标，代表最大的增量。一个完整的直线或者风杆代表一个较小的增量；半条直线表示最小的增量。

半直线、直线和三角形相应地增量依次为5、10和65。这里的值，至少对于气象学家，表示节每小时（knots）的风速。

我们把风杆从左向右排列依次表示的大小为：0、5、10、15、30、40、40、50、60和100节。这里每一个风杆的方向是相同的，为从北向南，因为每一个风杆的东西风速为0。

8.2.1 准备工作

风杆可以通过matplotlib中的matplotlib.pyplot.barbs函数生成。

barbs函数接受多种参数，主要应用在通过指定X和Y坐标来表示所

观测数据点的位置。第二对参数U、V，表示在北—南和东—西方向上以knots为单位的向量的大小。

其他一些有用的参数有中心点、大小和各种着色参数。

中心点（pivot）参数表示在网格点上显示的箭头的一部分。箭头可以围绕中心点旋转。箭头可以围绕其尖端或者中间旋转，这些值都是有效的中心点参数。

风杆由几部分组成，因此我们可以设置任何一部分的颜色。以下是几个与设置颜色有关的参数。

◆ barbcolor：定义了风杆中除旗标之外的所有部分的颜色。

◆ flagcolor：定义了风杆上任何旗标的颜色。

◆ facecolor：如果上面两个颜色参数都没有指定（或者使用rcParams 的默认值），则使用该参数。

如果指定了前两个参数中的任何一个，facecolor参数将被覆盖。facecolor参数常用于为多边形着色。

大小参数（sizes）指定了与风杆长度相关的属性的大小。这是一个系数的集合，可以通过以下任何一个或者所有的关键字指定。

◆ spacing：定义旗标/风杆属性间的间距。

◆ height：定义箭杆到旗标或者风杆顶部的距离。

◆ width：定义旗标的宽度。

◆ emptybarb：定义用于最小值的圆圈的半径。

8.2.2 操作步骤

让我们通过执行下面的步骤来演示如何使用barb函数。

1.生成一个坐标网格来模拟观测点。

2.模拟风速的观测值。

3.绘制风杆图。

4.绘制箭头来显示不同的外观。

下面是生成图表的代码：

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-20, 20, 8)
y = np.linspace( 0, 20, 8)
# make 2D coordinates
X, Y = np.meshgrid(x, y)
U, V = X+25, Y-35
# plot the barbs
plt.subplot(1,2,1)
plt.barbs(X, Y, U, V, flagcolor='green', alpha=0.75)
plt.grid(True, color='gray')
# compare that with quiver / arrows
plt.subplot(1,2,2)
plt.quiver(X, Y, U, V, facecolor='red', alpha=0.75)
# misc settings
plt.grid(True, color='grey')
plt.show()
```

以上代码生成如图8-2所示的两个图形。

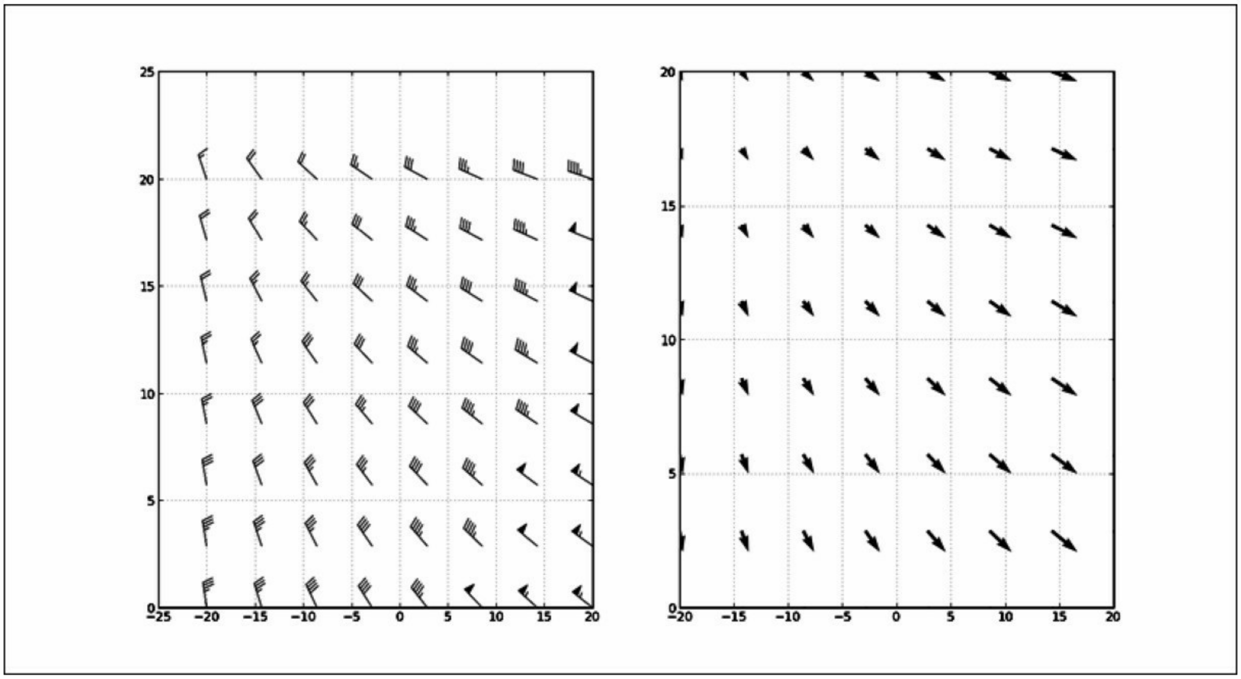


图8-2

8.2.3 工作原理

为了演示如何用相同的数据能呈现出不同信息，我们使用matplotlib中的风杆图和箭形图对模拟的风力观测数据分别可视化。

首先，用NumPy生成不同的x和y样本数组。然后，使用NumPy的meshgrid()函数创建出一个2D坐标网格，我们的观测数据是在该网格特定坐标上采样的。最后，U和V是以knots为单位的NS（北—南）和EW（东—西）方向的风速值。为了本节的演示需要，我们调整了已有的X和Y矩阵中的一些值。

然后，把图表分成两个子区，在左边的区域绘制风杆，在右边的区域绘制箭头补片。同时我们轻微地调整了两个子区的颜色和透明度，并且打开了两个子区的网格显示。

8.2.4 补充说明

这在北半球完全没有问题，因为在那里风是按照逆时针方向旋转的，并且羽毛（风杆的三角形，全直线和半直线）指向低压的方向。在南半球，情况就颠倒了，这时我们的风力风杆图就不能正确地表现要可视化的数据了。

我们必须反转羽毛的方向。幸运的是，barbs函数有一个参数flip_barb。这个参数可以是一个单一的布尔值（True或False），或者是一个与数据序列相同长度的布尔值序列，这时候序列中的每一个元素指定了每个风杆的倾斜方向。

8.3 绘制箱线图

你想在一幅图表中可视化一系列测量（或观测）数据来显示这些数据的属性（如中值、数据扩散和数据分布）吗？你想以一种可以直观地比较几个相似的数据系列的方式来可视化数据吗？你会怎样可视化它们呢？这是该用到箱线图的时候了！如果你在和一个习惯了密集信息的人讨论问题，箱线图很可能是进行分布比较最合适不过的图表类型了。

从比较学校间的测验成绩，到比较变化（优化）前后的流程参数，箱线图的用途很广。

8.3.1 准备工作

箱线图都由哪些元素组成？正如我们从图 8-3 中看到的，在箱线图中有几个非常重要的载有信息的元素。第一个是箱体，包含从低四分位到高四分位的四分位范围信息。数据的中值由横穿箱体的一条线段表示。

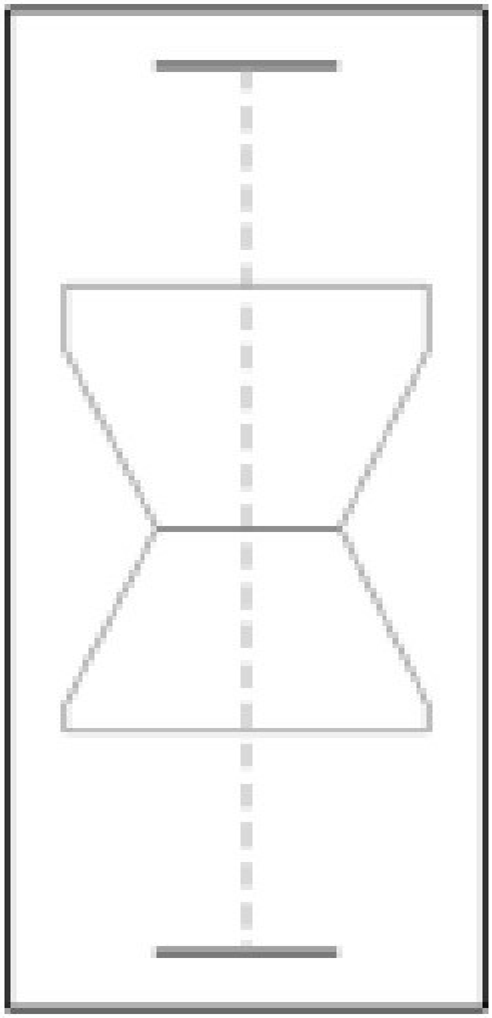


图8-3

箱须从数据的第一个四分位（25%）到最后一个四分位（75%），向箱体的两端延伸。换句话说，箱须从四分位间范围的基线开始向外延伸四分位间距的 1.5 倍。在正态分布的情况下，箱须将涵盖总数据范围的99.3%。

如果在箱须范围外还有值，它们将被显示为异常值[1]。否则，箱须将覆盖整个数据范围。

视情况而定，箱体也可以包含关于围绕中值的置信区间信息。这通过在箱体上的一个凹槽来表示。该信息可以用来指出两组数据是否有着相似的分布情况。然而，这并不严格，只是可以被人眼观测的一个指导

信息。

8.3.2 操作步骤

在接下来的小节中，我们将学习如何用matplotlib创建箱线图。我们将完成以下步骤。

1.采样一定量的过程数据，其中每一个整数值代表在观测的运行期间错误的发生率。

2.把PROCESSES字典的数据读入DATA。

3.把PROCESSES字典的标签读入LABELS。

4.用matplotlib.pyplot.boxplot绘制箱线图。

5.从图表中去掉一些图表垃圾信息（chartjunk）[2]。

6.添加坐标轴标签。

7.显示图表。

下面是实现这些步骤的代码。

```
import matplotlib.pyplot as plt
# define data
PROCESSES = {
    "A": [12, 15, 23, 24, 30, 31, 33, 36, 50, 73],
    "B": [6, 22, 26, 33, 35, 47, 54, 55, 62, 63],
    "C": [2, 3, 6, 8, 13, 14, 19, 23, 60, 69],
    "D": [1, 22, 36, 37, 45, 47, 48, 51, 52, 69],
    }
DATA = PROCESSES.values()
LABELS = PROCESSES.keys()
plt.boxplot(DATA, notch=False, widths=0.3)
# set ticklabel to process name
```

```
plt.gca().xaxis.set_ticklabels(LABELS)
# some clean up(removing chartjunk)
# turn the spine off
for spine in plt.gca().spines.values():
    spine.set_visible(False)
# turn all ticks for x-axis off
plt.gca().xaxis.set_ticks_position('none')
# leave left ticks for y-axis on
plt.gca().yaxis.set_ticks_position('left')
# set axes labels
plt.ylabel("Errors observed over defined period.")
plt.xlabel("Process observed over defined period.")
plt.show()
```

上面代码生成的图形如图8-4所示。

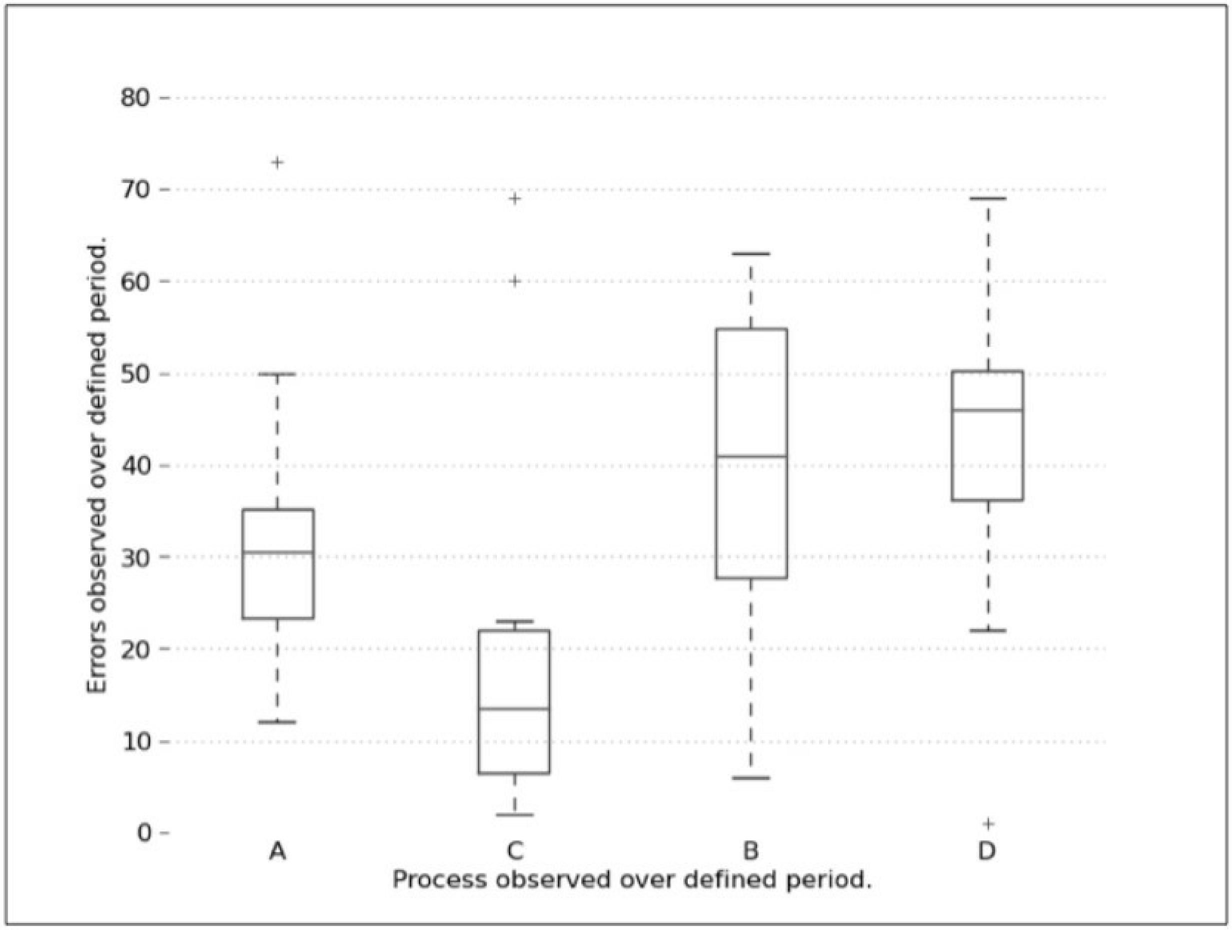


图8-4

8.3.3 工作原理

首先计算出给定数据DATA的四分位数，然后绘制出箱线图。

这些四分位数被用来计算绘制箱体和箱须所需的线段。

我们调整了图表使其看起来更美观，去掉了所有不必要的线条（指多余的线条，如“图表垃圾”，在 Edward R. Tufte 编写的著作（The Visual Display of Quantitative Information 一书中提到过）。这些线条不包含任何信息，却为读者徒增许多压力，让他们在发现真正有价值的信息之前花心思来理解这些所有的线条。

8.4 绘制甘特图

一种被广泛使用的基于时间数据的可视化方式是甘特图。甘特图由机械工程师 Henry Gantt在19世纪10年代发明，并以该工程师的名字命名，专门用来可视化项目管理中的工作分解结构。甘特图因具有很强的叙述性而深受管理者的喜爱，但是却不那么受雇员的喜爱，尤其是当临近项目截止日期的时候。

因为甘特图的使用非常普遍，即使我们为它添加了过多附加（相关的和不相关的）信息，每个人也都能够读懂它。

基本的甘特图在x轴上有一个时间序列，在y轴上有一些表示任务或者子任务的标签。任务持续时间通常被可视化为一条线段或者一个柱状图表，从给定任务的开始时间延伸到其结束时间。

如果存在子任务，一个或多个子任务有一个父任务，在这种情况下，任务的总时间是所有子任务的时间之和，在计算时，重叠时间和间隔时间都算在内。这在执行关键路径分析时非常有用。

关键路径分析是一种数学分析方法，它计算一条包含所有所需任务在内的路径，在计算时考虑任务间的相互依赖，这样就可以计算出项目从开始到完成的总时间。它在项目管理中是一个非常重要的工具，可以被普遍地应用于任何类型项目的时间和资源计划中。

因此，本节将学习如何用Python创建甘特图。

8.4.1 准备工作

有许多成熟的软件应用程序和服务可以用来创建灵活且复杂的甘特图。我们将试着向你展示如何在纯Python环境中，不依赖其他外部应用

程序的情况下，创建出美观并且信息丰富的甘特图。

示例中的甘特图不支持嵌套任务，但是它对于描述简单的任务分解结构已经够用了。

8.4.2 操作步骤

我们将使用下面的代码示例展示如何使用Python和matplotlib绘制甘特图。执行下面的步骤。

1.加载包含任务的TEST_DATA，并用TEST_DATA实例化Gantt类。

2.每一个任务包含一个标签，及开始和结束时间。

3.在坐标轴上绘制水平条来表示所有的任务。

4.为渲染的数据格式化x轴和y轴。

5.让图表布局紧凑些。

6.显示甘特图。

下面是示例代码。

```
from datetime import datetime
import sys
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager
import matplotlib.dates as mdates
import logging
class Gantt(object):
    '''
    Simple Gantt renderer.
    Uses *matplotlib* rendering capabilities.
```

```
'''
# Red Yellow Green diverging colormap
# from http://colorbrewer2.org/
RdYlGr = ['#d73027', '#f46d43', '#fdae61',
    '#fee08b', '#ffffbf', '#d9ef8b',
    '#a6d96a', '#66bd63', '#1a9850']
POS_START = 1.0
POS_STEP = 0.5
def __init__(self, tasks):
    self._fig = plt.figure()
    self._ax = self._fig.add_axes([0.1, 0.1, .75, .5])
    self.tasks = tasks[::-1]
def _format_date(self, date_string):
    '''
    Formats string representation of *date_string* into
    *matplotlib. dates*
    instance.
    '''
    try:
        date = datetime.strptime(date_string, '%Y-%m-%d
%H:%M:%S')
    except ValueError as err:
        logging.error("String '{0}' can not be converted to
        datetime object: {1}"
            .format(date_string, err))
        sys.exit(-1)
    mpl_date = mdates.date2num(date)
```

```
        return mpl_date
    def _plot_bars(self):
        '''
        Processes each task and adds *barh* to the current *self._ax* (*axes*).
        '''
        i = 0
        for task in self.tasks:
            start = self._format_date(task['start'])
            end = self._format_date(task['end'])
            bottom = (i * Gantt.POS_STEP) + Gantt.POS_START
            width = end - start
            self._ax.barh(bottom, width, left=start, height=0.3,
                align='center', label=task['label'],
                color = Gantt.RdYlGr[i])
            i += 1
    def _configure_yaxis(self):
        '''y axis'''
        task_labels = [t['label'] for t in self.tasks]
        pos = self._positions(len(task_labels))
        ylocs = self._ax.set_yticks(pos)
        ylabels = self._ax.set_yticklabels(task_labels)
        plt.setp(ylabels, size='medium')
    def _configure_xaxis(self):
        '''x axis'''
        # make x axis date axis
        self._ax.xaxis_date()
```

```
    # format date to ticks on every 7 days
    rule = mdates.rrulewrapper(mdates.DAILY, interval=7)
    loc = mdates.RRuleLocator(rule)
    formatter = mdates.DateFormatter("%d %b")
    self._ax.xaxis.set_major_locator(loc)
    self._ax.xaxis.set_major_formatter(formatter)
    xlabels = self._ax.get_xticklabels()
    plt.setp(xlabels, rotation=30, fontsize=9)
def _configure_figure(self):
    self._configure_xaxis()
    self._configure_yaxis()
    self._ax.grid(True, color='gray')
    self._set_legend()
    self._fig.autofmt_xdate()
def _set_legend(self):
    '''
    Tweak font to be small and place *legend*
    in the upper right corner of the figure
    '''
    font = font_manager.FontProperties(size='small')
    self._ax.legend(loc='upper right', prop=font)
def _positions(self, count):
    '''
    For given *count* number of positions, get array for the
positions.
    '''
    end = count * Gantt.POS_STEP + Gantt.POS_START
```

```
        pos = np.arange(Gantt.POS_START, end, Gantt.POS_STEP)
        return pos
```

下面的代码定义了生成甘特图的主函数。在这个函数中，我们把数据加载到一个实例中，绘制出相应的水平条、设置好时间坐标轴（x 轴）的日期格式，并设置 y 轴（项目任务）上的值。

```
    def show(self):
        self._plot_bars()
        self._configure_figure()
        plt.show()
    if __name__ == '__main__':
        TEST_DATA = (
            { 'label': 'Research',        'start':'2013-10-01
            12:00:00', 'end': '2013-10-02 18:00:00'}, # @IgnorePep8
            { 'label': 'Compilation',    'start':'2013-10-02
            09:00:00', 'end': '2013-10-02 12:00:00'}, # @IgnorePep8
            { 'label': 'Meeting #1',    'start':'2013-10-03
            12:00:00', 'end': '2013-10-03 18:00:00'}, # @IgnorePep8
            { 'label': 'Design',        'start':'2013-10-04
            09:00:00', 'end': '2013-10-10 13:00:00'}, # @IgnorePep8
            { 'label': 'Meeting #2',    'start':'2013-10-11
            09:00:00', 'end': '2013-10-11 13:00:00'}, # @IgnorePep8
            { 'label': 'Implementation', 'start':'2013-10-12
            09:00:00', 'end': '2013-10-22 13:00:00'}, # @IgnorePep8
            { 'label': 'Demo',        'start':'2013-10-23
            09:00:00', 'end': '2013-10-23 13:00:00'}, # @IgnorePep8
            )
        gantt = Gantt(TEST_DATA)
```

```
    gantt.show()
```

代码将生成一个简单美观的甘特图，如图8-5所示。

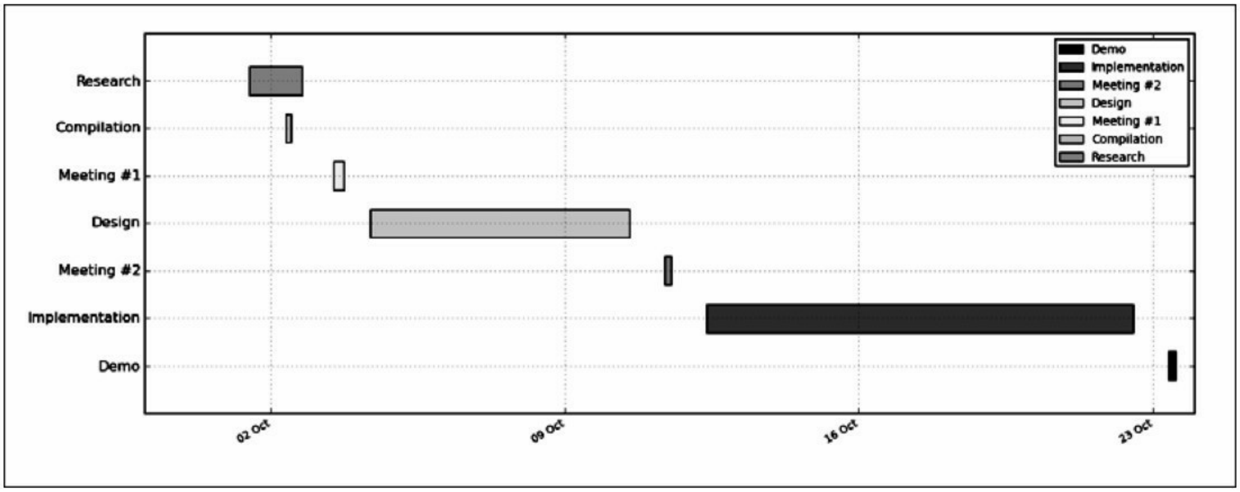


图8-5

8.4.3 工作原理

我们从上面代码底部的"__main__"中 if 语句检查之后开始读。在给定 TEST_DATA参数实例化 Gantt 类之后，我们为该实例创建一些必要的字段。把 TASK_DATA[3]保存在self.tasks字段中，并且创建坐标轴和图形窗口来保存接下来要创建的图表。

然后，在实例上调用show()方法，该方法执行所需的步骤创建出甘特图。

```
def show(self):
    self._plot_bars()
    self._configure_figure()
    plt.show()
```

绘制水平条需要一个循环，在循环中把每一个任务的名称和持续时间数据应用到matplotlib.pyplot.barh 函数上，并把它添加到 self._ax 坐标轴中。通过给每一个任务一个不同（增量）的bottom参数值，我们可以

把每个任务放在一个单独的通道上。

并且，为了能容易地把任务映射到它们的名字上，我们对其循环应用colorbrewer2. org工具生成的divergent颜色表。

下一步是配置图表，即设置 x 轴上的日期格式和 y 轴上的刻度位置和标签，来与用matplotlib.pyplot.barh函数绘制的任务进行匹配。

然后，对grid和legend做最后的调整。

最后，调用plt.show()把图表显示出来。

8.5 绘制误差条

误差条在显示图表中数据的离散度时非常有用。作为可视化的一种形式，它们相对比较简单；然而，它们也有一些问题，因为在不同的学科和出版物中，把什么作为错误来显示是不同的。但这并没有减少误差条的有效性，只是要求我们加倍小心，并且要明确地表述可视化误差条的误差性质。

8.5.1 准备工作

为了能在裸观测数据上绘制误差条，需要计算所要显示数据的平均值和误差。

我们计算的误差表示的是从观测得出数据的平均值的95%置信区间。该平均值是稳定的，指对整个总体观测的良好估计。

matplotlib通过matplotlib.pyplot.errorbar函数来支持该类型的图表。

它提供了不同种类的误差条。误差条可以是竖直的（yerr）或者水平的（xerr），对称的或者非对称的。

8.5.2 操作步骤

在下面的代码中我们将进行以下操作。

1.使用一些包含四组观测值的采样数据。

2.对每一组观测值，计算出平均值。

3.对每一组观测值，计算出95%置信区间。

4.使用竖直对称的误差条绘制出误差条图。

代码如下。

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as sc
TEST_DATA = np.array([[1,2,3,2,1,2,3,4,2,3,2,1,2,3,4,4,3,2,3,2,3,2,1],
    [5,6,5,4,5,6,7,7,6,7,7,2,8,7,6,5,5,6,7,7,7,6,5],
    [9,8,7,8,8,7,4,6,6,5,4,3,2,2,2,3,3,4,5,5,5,6,1],
    [3,2,3,2,2,2,2,3,3,3,3,4,4,4,4,5,6,6,7,8,9,8,5],
    ])
# find mean for each of our observations
y = np.mean(TEST_DATA, axis=1, dtype=np.float64)
# and the 95% confidence interval
ci95 = np.abs(y - 1.96 * sc.sem(TEST_DATA, axis=1))
# each set is one try
tries = np.arange(0, len(y), 1.0)
# tweak grid and setup labels, limits
plt.grid(True, alpha=0.5)
plt.gca().set_xlabel('Observation #')
plt.gca().set_ylabel('Mean (+- 95% CI)')
plt.title("Observations with corresponding 95% CI as error bar.")
plt.bar(tries, y, align='center', alpha=0.2)
plt.errorbar(tries, y, yerr=ci95, fmt=None)
plt.show()
```

上述代码将生成带误差条的图形，该图形显示的95%置信区间为沿y轴方向延伸的须线。记住，须线越宽，表示观测的平均值为真的可能性就越低。图8-6所示为上述代码的输出。

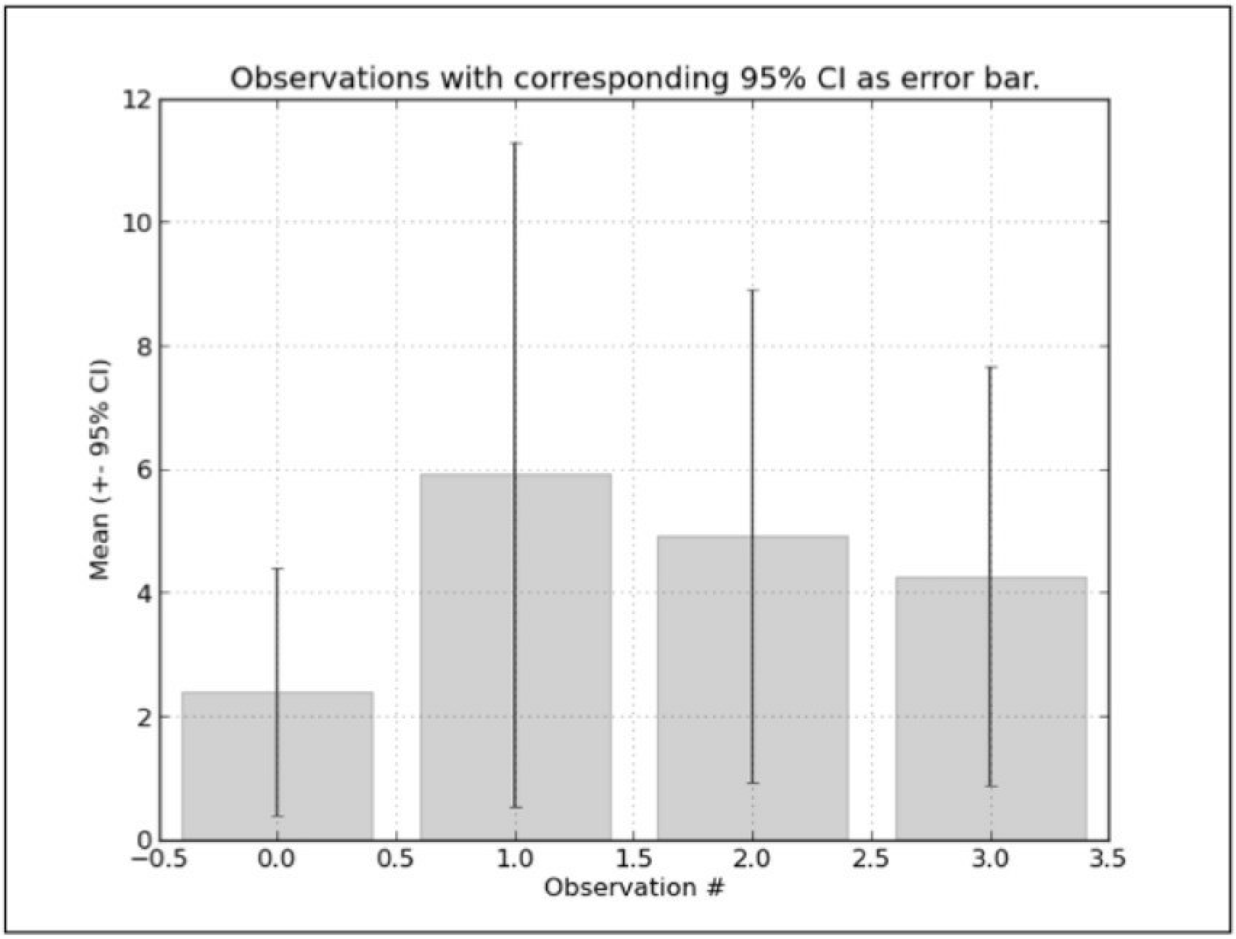


图8-6

8.5.3 工作原理

为了避免在每一个观测数据集合上进行迭代，我们使用NumPy的向量化方法来计算均值和标准误差，然后用计算得出的值绘制和计算误差值。

NumPy的向量化实现是用C语言编写的（在Python中被调用），这能让计算提速好几个数量级。

这对于少量数据点并不十分重要，但是对于成千上万个数据点来说，NumPy的向量化实现却能在关键时刻帮我们创建出响应式的应用程序。

此外，你可能注意到，我们在np.mean函数调用中显式地指定了dtype=np.float64。按照NumPy官方文档（http://docs.scipy.org/doc/numpy/reference/generated/ numpy.mean.html）的解释，如果使用单精度，np.mean可能不准确，最好使用np.float32来计算均值。如果对于你的机器来说性能不是问题，则使用np.float64。

8.5.4 补充说明

对于在误差条上要显示什么的讨论从未停息。一些人建议使用SD、2SD、SE 或者95%CI。我们必须要明白这些值之间的区别以及它们的用途，才能对什么时候使用哪种值做出合理的解释。

标准偏差（Standard Deviation）描述的是单个数据点围绕平均值的分布情况。如果有一个正态分布，那么我们知道68.2%（～2/3）的数据值将落在±SD之间，95.4%的值将落在±2*SD之间。

标准误差（Standard Error）通过 SD 除以 N 的平方根（SD/√N）计算得出，其中 N 为数据点的数量。如果我们能够进行多次相同的采样（如进行上百次相同的研究），标准误差（SE）描述的是平均值的变动程度。

置信区间从SE计算得出，与通过标准误差计算得出值范围的方式类似。为了计算95%置信区间，必须在平均值上加/减 1.96*SE，或者使用公式 95% CI = M ± (1.96 * SE)。置信区间越宽，我们的估计正确的可能性越小。

我们看到，为了确保我们的估计是正确的，并且给读者提供出证据，应该把置信区间显示出来，置信区间携带了标准误差的信息。如果置信区间很小，证明平均值是稳定的。

8.6 使用文本和字体属性

我们已经学习了如何通过添加图例来对图表进行注解，但是有时候，我们需要添加更多的文本信息。本节将解释和演示matplotlib中更多文本操作的特性，为更高级的排版需要提供一个强大的工具箱。

在本节中我们不介绍LaTex，因为本章有“用LaTex渲染文本”一节对其进行专门介绍。

8.6.1 准备工作

我们首先列出 matplotlib 提供的最有用的一系列函数。这些函数中的大多数都能在pyplot 模块的接口中找到，但是我们在这里列出它们最初的函数。如果某个特定的文本特性在本节没有涉及的话，你能够借助它们去了解更多内容。

表 8-1 显示的是基本的文本操作，以及它们在 matplotlib OO API 中对应的函数。

表8-1

matplotlib.pyplot	Matplotlib API	描　　述
`text`	`matplotlib.axes.Axes.text`	在指定的位置（x，y）为坐标轴添加文本。`fontdict` 参数允许我们覆盖一般的字体属性，或者可以使用 `kwargs` 覆盖特定的属性
`xlabel`	`matplotlib.axes.Axes.set_xlabel`	设置 x 轴的标签。通过 `labelpad` 指定标签和 x 坐标轴之间的间隔

续表

matplotlib.pyplot	Matplotlib API	描　　述
ylabel	matplotlib.axes.Axes.set_ylabel	和 xlabel 类似，但用于 y 轴
title	matplotlib.axes.Axes.set_title	设置坐标轴的标题。接受所有一般的文本属性，如 fontdict 和 kwargs
suptitle	matplotlib.figure.Figure.suptitle	为图表添加一个居中的标题。通过 kwargs 接受所有通用文本属性。使用 Figure 坐标
figtext	matplotlib.figure.Figure.text	在图表的任意位置添加文本。位置通过 x、y 定义，使用图表的归一化坐标。使用 fontdict 覆盖字体属性，但也支持使用 kwargs 覆盖任何文本相关的属性

在窗口或者数据坐标中用于绘制和保存文本的基类是matplotlib.text.Text类。它支持对文本对象位置进行设定，以及一系列属性的设置，用来调整图表或窗口中字符串的显示效果。

matplotlib.text.Text实例支持的字体属性如表8-2所示。

表8-2

属　　性	值	描　　述
family	'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'	指定字体名称或字体类型。如果是一个列表，那么按优先级顺序排列，这样将使用第一个匹配的字体名称
size 或 fontsize	12, 10,... or 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'	指定字体的相对大小或者绝对点数，或者指定字体的相对大小为一个大小字符串
style 或 fontstyle	'normal', 'italic', 'oblique'	指定字体风格为一个字符串

续表

属　　性	值	描　　述
variant	'normal', 'small-caps'	指定字体的变体形式
weight或fontweight	0-1000 or 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extrabold', 'black'	指定字体粗细或者使用一个特定的粗细字符串。字体粗细定义为相对于字体高度的字符轮廓厚度
stretch或fontstretch	0-1000 or 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded', 'ultra-expanded'	指定字体的拉伸。拉伸定义为水平的压缩或者扩张。该属性目前没有实现
fontproperties		默认使用matplotlib.font_manager. Font Properties 实例。该类存储并管理 W3C CSS Level1 规范中描述的字体属性。规范网址为 http://www.w3.org/ TR/1998/REC-CSS2- 19980512/

我们也可以指定包含文本的背景框，并可以为该背景框指定颜色、边界和透明度。

基本的字体颜色从rcParams['text.color']中读取，当然是如果在当前的实例上没有指定字体颜色的前提下。

指定的字体也可以按照视觉的需要进行对齐。对齐属性如下。

◆ horizontalalignment 或 ha：允许的字体水平对齐方式有 center、left和right。

◆ verticalalignment 或 va：允许的值有 center、top、bottom和baseline。

◆ multialignment：允许跨多行的文本字符串对齐，允许的值有left、right和center。

8.6.2 操作步骤

到目前一切顺利，但是很难可视化我们能够创建的字体的所有变体。因此在这里先说明一下我们可以做的事情。在下面的代码中我们将执行以下步骤。

1.列出我们想要改变的字体的所有可能的属性。

2.在第一个变体集合上循环：字体类型和大小。

3.在第二个变体集合上循环：字体粗细和风格。

4.为两个变体渲染文本示例，并在图表上以文本的形式打印出变体组合。

5.从图表中去掉坐标轴，因为它们毫无用处。

下面是代码：

```
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
# properties:
families = ['serif', 'sans-serif', 'cursive', 'fantasy', 'monospace']
sizes = ['xx-small', 'x-small', 'small', 'medium', 'large',
```

```
    'x-large', 'xx-large']
styles = ['normal', 'italic', 'oblique']
weights = ['light', 'normal', 'medium', 'semibold', 'bold', 'heavy',
'black']
variants = ['normal', 'small-caps']
fig = plt.figure(figsize=(9,17))
ax = fig.add_subplot(111)
ax.set_xlim(0,9)
ax.set_ylim(0,17)
    # VAR: FAMILY, SIZE
y = 0
size = sizes[0]
style = styles[0]
weight = weights[0]
variant = variants[0]
for family in families:
    x = 0
    y = y + .5
    for size in sizes:
        y = y + .4
        sample = family + " " + size
        ax.text(x, y, sample, family=family, size=size,
            style=style, weight=weight, variant=variant)
# VAR: STYLE, WEIGHT
y = 0
family = families[0]
size = sizes[4]
```

```
variant = variants[0]
for weight in weights:
    x = 5
    y = y + .5
    for style in styles:
        y = y + .4
        sample = weight + " " + style
        ax.text(x, y, sample, family=family, size=size,
            style=style, weight=weight, variant=variant)
ax.set_axis_off()
plt.show()
```

上述代码将生成如图8-7所示的截图。


monospace xx-large
monospace x-large
monospace large
monospace medium
monospace small
monospace x-small
monospace xx-small
fantasy xx-large
fantasy x-large
fantasy large
fantasy medium
fantasy small
fantasy x-small
fantasy xx-small
cursive xx-large
cursive x-large
cursive large
cursive medium
cursive small
cursive x-small
cursive xx-small
sans-serif xx-large
sans-serif x-large
sans-serif large
sans-serif medium
sans-serif small
sans-serif x-small
sans-serif xx-small
serif xx-large
serif x-large
serif large
serif medium
serif small
serif x-small
serif xx-small
black oblique
black italic
black normal
heavy oblique
heavy italic
heavy normal
bold oblique
bold italic
bold normal
semibold oblique
semibold italic
semibold normal
medium oblique
medium italic
medium normal
normal oblique
normal italic
normal normal
light oblique
light italic
light normal

图8-7

8.6.3 工作原理

代码非常直白易懂，因为我们只是在属性元组上循环两次就把它们的值打印了出来。

这里采用的唯一技巧是设置图表画布上字体的位置，因为它让我们有了一个布局良好的文本示例，并可以很容易地进行比较。

记住，matplotlib使用的默认字体取决于你所运行的操作系统，因此以上截图可能看起来会稍微有所不同。这个截图是使用标准 Ubuntu 13.04 预装的字体渲染出来的。

8.7 用LaTeX渲染文本

如果想要绘制更多的科学图形并解释数学应用，由于它们会在图表中使用科学符号和复杂的公式，我们需要对此有更好的支持。

虽然 matplotlib 支持数学文本渲染，但是对其最佳的支持来自 LaTex 社区，并且在实践中已经得到多年的印证。

LaTeX 是一个用于生成科学技术文档的高质量的排版系统，已经是事实上的科学排版或出版物的标准。它是一个免费的软件，在当今使用的大多数桌面系统上都可以通过预打包的二进制安装文件得到它。因此，它的安装非常简单。

LaTeX 的基本语法与标记语言相似，因此要生成满意的内容，我们需要集中在编写结构而不是处理外观和风格上。例如：

```
\documentclass{article}
\title{This here is a title of my document}
\author{Peter J. S. Smith}
\date{September 2013}
\begin{document}
  \maketitle
  Hello world, from LaTeX!
\end{document}
```

我们看到这与常用的文本编辑器不同，常用的文本编辑器拥有一个 WYSIWYG[4]编辑环境，风格已经被应用到了文本中。这样有时候很好，但是对于科学出版物，风格是次要考虑的问题；主要的关注点是得到恰当、正确和有效的内容。这里的内容指的是数学符号（通常有很多），还包括图形。

除此之外，还有更多地特性如自动生成目录和索引，这对于大中型的出版物是非常重要的。这些是LaTeX系统的主要关注点。

因为本书不是关于LaTeX的书籍，我们就在此做个快速的介绍。更多的文档可以从其项目的网站获得，网址为http://latex-project.org/。

8.7.1 准备工作

在开始演示matplotlib对于使用LaTeX渲染文本的支持之前，需要在我们的系统上安装以下包。

◆ LaTeX system：最常用的一个是 TeX Live 预打包发行版本。

◆ DVI to PNG converter：通过生成抗锯齿的屏幕分辨率图像，它把从 TeX 获得的DVI文件生成PNG图形。

◆ Ghost script：这是必需的，除非已经通过 TeX Live 发行包安装了该包。

对于不同的操作系统有不同的LaTeX环境的预打包系统。对于基于Linux的系统，TeX Live 是一个完整的 TeX 系统；对于 Mac OS，推荐的环境是 MacTeX 发行包；对于 Windows环境，proTeX系统将会安装所有的TeX支持，包括LaTeX。

不管安装了哪个包，请确保它已经包含字体库、排版和预览程序，以及不同语言的TeX文档。

我们将为Linxu系统安装用于Ubuntu的textlive和dvipng包。可以用下面的命令来安装。

```
$ sudo apt-get install texlive dvipng
```

下一步设置text.usetex为True，告诉matplotlib使用LaTeX。我们可以在自定义的.matplotlibrc中通过设置rcParams['text']来完成，该文件位于用户主目录（在基于 Unix 的系统中为/home/<user>/.matplotlibrc ，在 Windows 系统下位于C:\Documents and Settings\<user>\. matplotlibrc），

或者通过使用以下代码来实现：

```
matplotlib.pyplot.rc('text', usetex=True)
```

代码的开始部分将告诉matplotlib，对于所有的文本渲染使用LaTeX。在添加任何图形和坐标轴之前进行此设置是非常重要的。

并不是所有的后端都支持LaTeX渲染，只有Agg、PS和PDF后端支持通过LaTeX渲染文本。

8.7.2 操作步骤

本小节演示一下LaTeX基本属性的用法，步骤如下。

1.生成一些样本数据。

2.对于当前绘图session，设置matplotlib使用LaTeX。

3.设置要使用的字体和字体属性。

4.写出等式语法。

5.演示希腊符号语法的用法。

6.绘制分数和分形的数学符号。

7.写出一些极限和指数表达式。

8.写出可能的范围表达式。

9.写出带文本和格式化文本的表达式。

10.在x轴和y轴标签上写出一些数学表达式作为图表的标题。

下面是执行这些步骤的代码。

```
import numpy as np
import matplotlib.pyplot as plt
# Example data
t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(4 * np.pi * t) * np.sin(np.pi*t/4) + 2
plt.rc('text', usetex=True)
```

```
plt.rc('font',**{'family':'sans-serif','sans-serif':['Helvetica'],
'size':16})
plt.plot(t, s, alpha=0.25)
# first, the equation for 's'
# note the usage of Python's raw strings
plt.annotate(r'$\cos(4 \times \pi \times {t}) \times \sin(\pi \times \frac{t}
4) + 2$', xy=(.9,2.2), xytext=(.5, 2.6), color='red', arrowprops=
{'arrowstyle':'->'})
# some math alphabet
plt.text(.01, 2.7, r'$\alpha, \beta, \gamma, \Gamma, \pi, \Pi, \phi, \varphi,
\Phi$')
# some equation
plt.text(.01, 2.5, r'some equations $\frac{n!}{k!(n-k)!}={n \choose k}$')
# more equations
plt.text(.01, 2.3, r'EQ1 $\lim_{x \to \infty} \exp(-x) = 0$')
# some ranges...
plt.text(.01, 2.1, r'Ranges: $( a ), [ b ], \{ c \}, | d |, \| e \|, \langle f
\rangle, \lfloor g \rfloor, \lceil h \rceil$')
# you can multiply apples and oranges
plt.text(.01, 1.9, r'Text:$50 apples \times 100 oranges = lots of juice$')
plt.text(.01, 1.7, r'More text formatting:$50 \textrm{ apples}\times
100\textbf{ apples} = \textit{lots of juice}$')
plt.text(.01, 1.5, r'Some indexing: $\beta = (\beta_1,\beta_2,\dotsc,
\beta_n)$')
# we can also write on labels
plt.xlabel(r'\textbf{time} (s)')
plt.ylabel(r'\textit{y values} (W)')
```

```
# and write titles using LaTeX
plt.title(r"\TeX\ is Number "
    r"$\displaystyle\sum_{n=1}^\infty\frac{-e^{i\pi}}{2^n}$!",
    fontsize=16, color='gray')
# Make room for the ridiculously large title.
plt.subplots_adjust(top=0.8)
plt.savefig('tex_demo')
plt.show()
```

上述代码将渲染出如图8-8所示的有大量文本的图表来展示LaTeX渲染的效果。

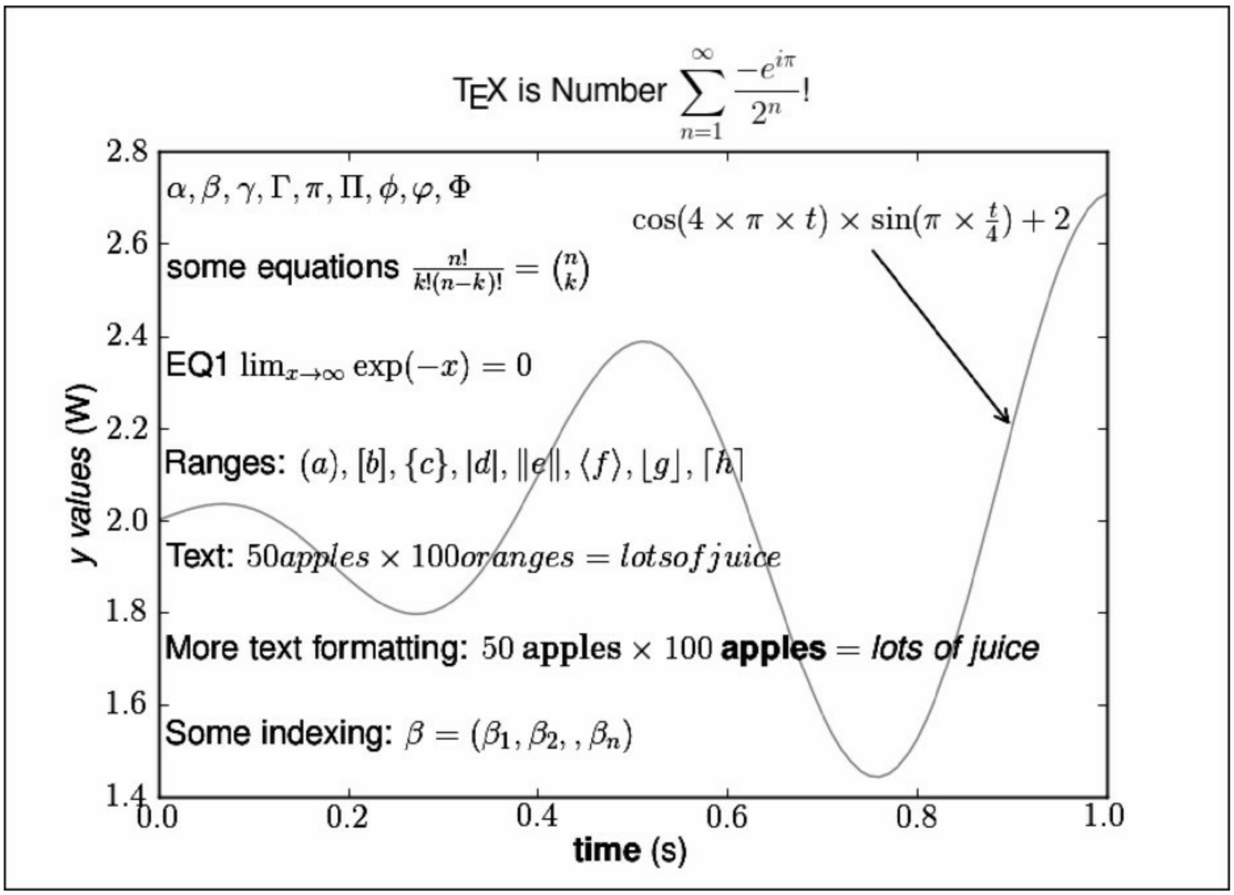


图8-8

8.7.3 工作原理

在设置完渲染引擎和字体属性之后，我们基本上使用了标准matplotlib函数调用来渲染文本，如matplotlib.pyplot.annotate、matplotlib.pyplot.text、matplotlib.pyplot.xlabel、matplotlib.pyplot.ylabel和matplotlib.pyplot. title。

不同的是，所有的字符串都是所谓的原始字符串，表明Python不会解释它们，不会发生字符串替换，因此LaTeX引擎将接收到和给定字符串完全相同的值。

TeX语法以及如何在matplotlib中使用该语法的更多示例可以从matplotlib官方文档获得，网址为http://matplotlib.org/users/mathtext.html#writing-mathematical-expressions。

注意，这个URL不是LaTeX的网址，而是matplotlib自身集成的TeX分析器的网址。这个分析器几乎支持和LaTeX相同的语法，完全能满足你的需要。

8.7.4 补充说明

如果在设置环境时遇到问题，或者有字体方面的各种问题，比如要么看起来很丑，要么不能得到LaTeX渲染效果，请确保你已经安装了所有所需的包，$PATH环境变量（如果在Windows系统上）已经设置为包含所有所需的二进制包，并且已经设置matplotlib为使用LaTeX来做文本渲染。

如果按照所有给定的指令还是不能得到预期结果，请参考matplotlib 官方网站http://matplotlib.org/users/usetex.html#possible-hangups和 LaTeX 社区网站 http://tex.stackex change. com/得到进一步的帮助。

众所周知，设置不会像预期的那样一帆风顺，各种怪事都可能发生。

8.8 理解pyplot和OO API的不同

在本节中，我们将试着解释一些matplotlib中的编程接口，对pyplot和面向对象的API （应用程序接口）做一个比较。了解这些后，我们就能根据手头的任务来决定为什么以及适合使用哪种接口。

8.8.1 准备工作

开始时matplotlib库和许多开源项目相似——对于作者面临的问题没有合适（免费）的解决方案，因此作者写了一个。MATLAB®面临的问题体现在处理手头工作的性能上（http://www.aosabook.org/en/matplotlib.html）。并且原作者已经具备了MATLAB®和 Python的知识，因此作者动手编写了matplotlib作为他当前项目需求的解决方案。

这就是 matplotlib 有一个类 MATLAB®的接口的主要原因。它让人们能够快速地绘制数据，而不用担心后台细节，比如matplotlib当前运行在什么平台上，底层使用的渲染库是哪个（是Linux下的GTK、QT、Tk的，或者Linux或Windows下地wxWidgets），或者我们是否借助 Cocoa 工具包在 Mac OS 系统上运行。所有这些都隐藏在 matplotlib 内部，在其之上是一个 matplotlib.pyplot 模块中的良好程序接口，这个有状态的接口处理创建图表和坐标轴的逻辑，并把它们与配置的后端联系起来。它也为当前图表和坐标轴保存了数据结构，可以通过plot命令进行调用。

matplotlib.pyplot就是我们本书大部分章节中用到的接口，它简单、直接，能胜任我们想要解决的大多数任务。matplotlib库就是在这种哲学

思想下设计出来的。我们必须能够在画图时使用尽可能少的命令，甚至只需一个命令（例如 plt.plot([1,2,3,4, 5]);plt.show()！）完成对于这些任务，我们不想被迫去思考关于对象、实例、方法、属性、渲染后端、图表、画布、线条和一些其他的图形元素。

如果你是从开头阅读本书，很可能已经注意到一些类在许多示例中都出现过比如FontProperties或者AxesGrid，此时我们需要matplotlib.pyplot模块提供的功能之外更多的功能。

面向对象的编程接口实现了所有隐藏的棘手工作，如渲染图形元素，把它们渲染到平台的图形工具上，以及处理用户的输入（鼠标和键盘点击）。没有什么阻止我们使用 OO API，而且这也是我们接下来要介绍的。

因此，如果把matplotlib看做一个软件，它由以下三部分组成。

◆ matplotlib.pylab 接口：这是用户用来创建类似 MATLAB®中的图形的一组函数。

◆ matplotlib API（也称为 matplotlib 前端）：这是用于创建和管理图表、文本、线条、图形等的一组类。

◆ 后端：这些是绘图驱动，它们把前台的抽象表示转换成一个文件或一个显示设备。

后端层包含了抽象接口类的具体实现。包含的类有FigureCanvas（画到纸上的一个表面）、Renderer（在画布上绘图的画笔）和Event（处理用户键盘点击和鼠标事件的类）。

代码也是分离的。抽象基类在 matplotlib.backend_bases 中，每一个具体实现在一个单独的模块中。例如，GTK 3后端在matplotlib.backends.backend_gkt3agg[5]中。

在这个体系中，有一个 Artist 类层级结构，很多棘手的工作都是在这里完成的。Artist知道Renderer的存在以及如何使用它在FigureCanvas

上绘制图像。大多数我们感兴趣的东西（文本、线条、刻度、刻度标签、图像等等）都是Artist类或者Artist类的子类（位于matplotlib.artist模块）。

matplotlib.artist.Artist 类包含了所有其子类共享的属性：坐标转化、剪切区、标签、用户事件处理器和可见性，结构如图8-9所示。

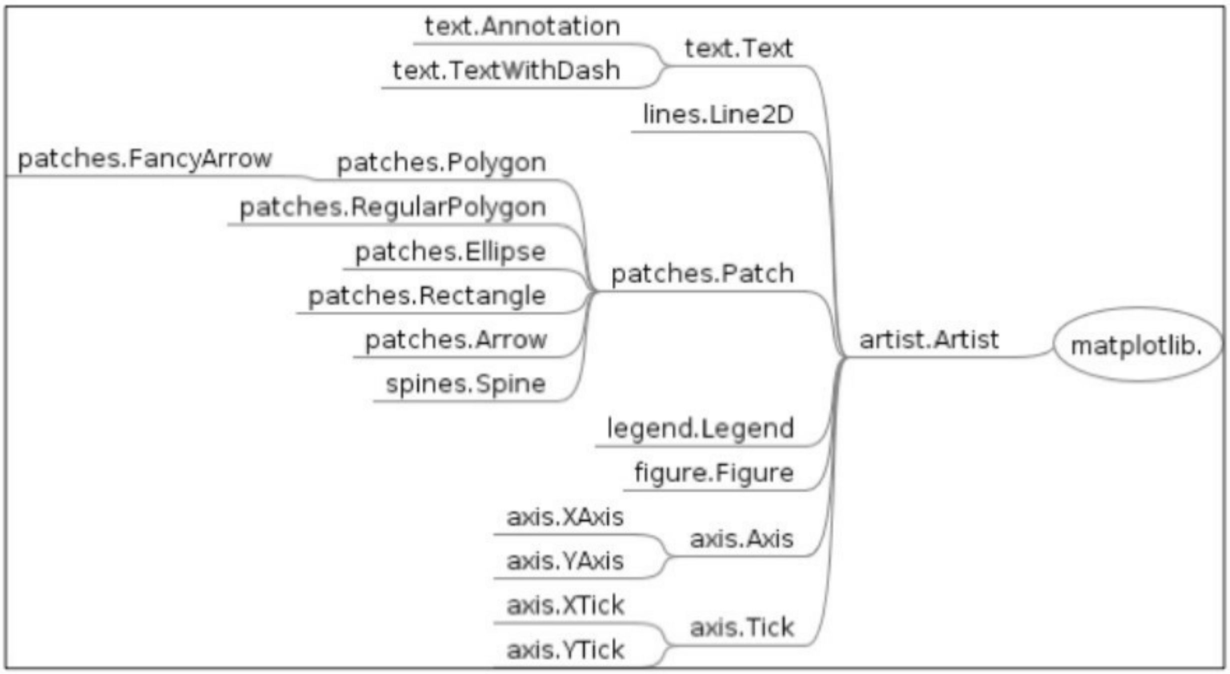


图8-9

在这个图表中，Artist 是大多数其他类的基类。有两个基本类别的类继承自Artist。第一类是简单类型的 artists，是一些可见的对象如Line2D、Rectangle、Circle和Text。第二类是组合的artists，是其他Artists的组合如Axis、Tick、Axes和Figure。例如，Figure有简单的artist——Rectangle作为背景，但还包含至少一个组合artist——Axes。

大部分绘图操作发生在Axes类（matplotlib.axes.Axes）上。图表背景元素如刻度、坐标轴线，以及背景补片的网格和颜色都包含在Axes中。Axes另一个重要的特性是，所有的helper方法创建其他简单类型artist，并把它们添加到Axes实例中，例如plot、hist和imshow。

举个例子，Axes.hist 创建了许多 matplotlib.patch.Rectangle 实例，

并把它们保存在Axes.patches集合中。

Axes.plot创建一个或多个matplotlib.lines.Line2D实例，并把它们保存在Axes.lines集合中。

8.8.2 操作步骤

我们将举个例子说明一下。

1.实例化一个用于自定义绘图的 matplotlib Path 对象。

2.创建对象的顶点。

3.创建路径的指令代码把这些顶点连接起来。

4.创建一个补片。

5.把它添加到figure的Axes实例中。

下面是实现代码。

```
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches
# add figure and axes
fig = plt.figure()
ax = fig.add_subplot(111)
coords = [
    (1., 0.), # start position
    (0., 1.),
    (0., 2.), # left side
    (1., 3.),
    (2., 3.),
    (3., 2.), # top right corner
    (3., 1.), # right side
```

```
    (2., 0.),
    (0., 0.), # ignored
    ]
line_cmds = [Path.MOVETO,
    Path.LINETO,
    Path.LINETO,
    Path.LINETO,
    Path.LINETO,
    Path.LINETO,
    Path.LINETO,
    Path.LINETO,
    Path.CLOSEPOLY,
    ]
# construct path
path = Path(coords, line_cmds)
# construct path patch
patch = patches.PathPatch(path, lw=1,
    facecolor='#A1D99B', edgecolor='#31A354')
# add it to *ax* axes
ax.add_patch(patch)
ax.text(1.1, 1.4, 'Python', fontsize=24)
ax.set_xlim(-1, 4)
ax.set_ylim(-1, 4)
plt.show()
```

上述代码生成的图形如图8-10所示。

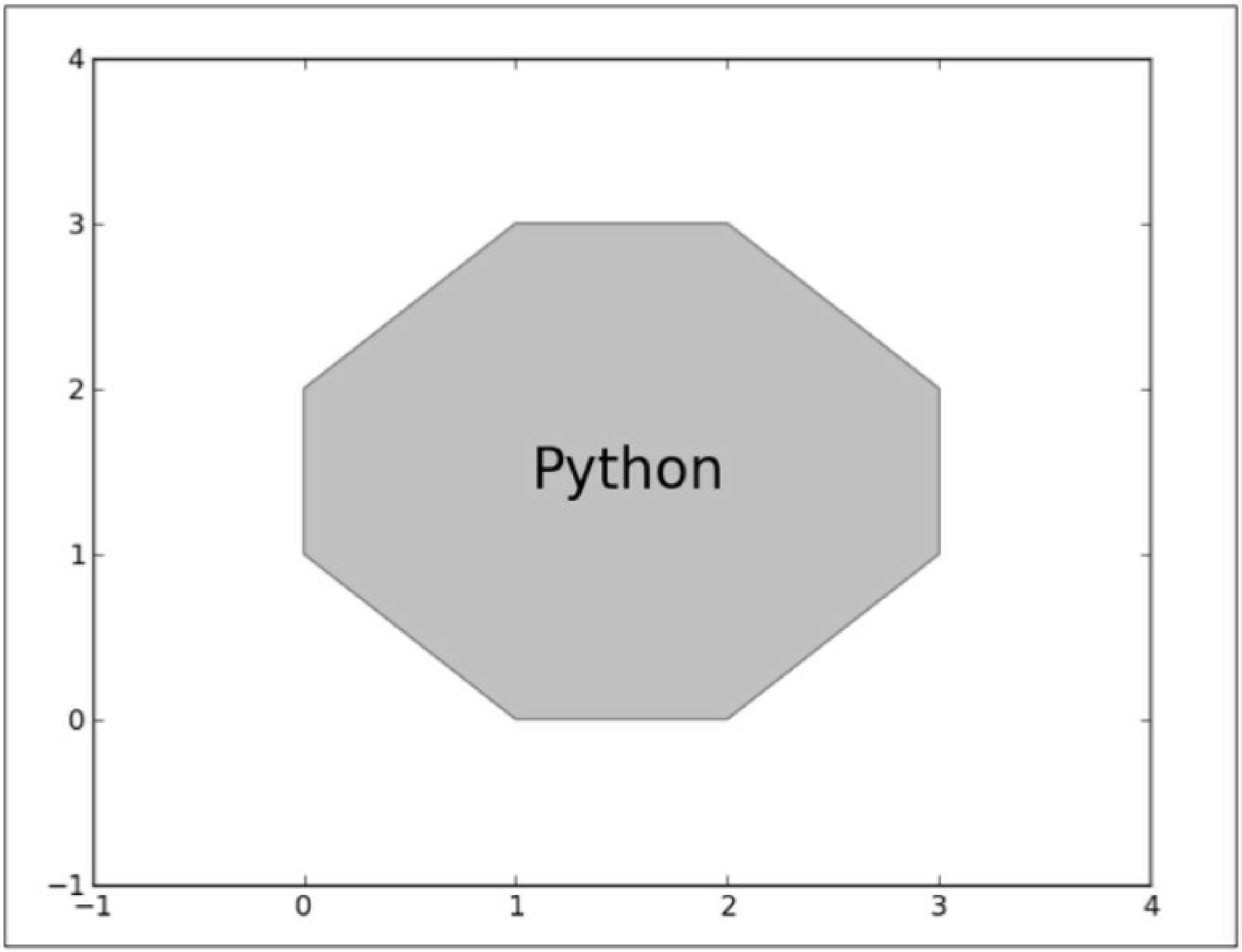


图8-10

8.8.3 工作原理

绘制这个八边形，我们使用了基本的补片类matplotlib.path.Path，它包含了绘制线条和曲线的基元的基本集合（moveto和lineto）。这些可以被用来绘制简单的多边形，也可以使用贝塞尔曲线绘制更高级的多边形。

首先，我们在数据坐标中指定了一组坐标，并为其匹配了一组在这些坐标上（或者顶点，如果你想这么称呼的话）执行的路径命令。对此我们实例化了一个 matplotlib. path.Path 对象。然后，用这个 path 创建了 matplotlib.patched. PathPatch 实例对象——一个普通的多重曲线路径补

片。

现在，可以把这个补片添加到图表的坐标轴（fig.axes集合）中，并且可以渲染图表来把多边形显示出来。

在这个例子中我们不想直接使用matplotlib.figure.Figure类来代替matplotlib.　pyplot.figure()调用。这样做的原因是pyplot.figure()在后台做了很多工作，比如从matplotlibrc文件读取rc参数（加载默认figsize、dpi和图表颜色设置），设置图表管理类（Gcf）等。我们可以手工做所有这些工作，但在我们知道真正要做什么之前，推荐的方式是通过pyplot.figure()创建图表。

作为一般经验法则，除非我们用pyplot接口无法完成某项工作，否则不应该接触如Figure、Axes和Axis的直接类，因为在后台进行着许多状态管理工作。因此，除非我们在开发matplotlib，否则应该避免打搅它们。

8.8.4 补充说明

如果你想进行互动和探索编程，最好通过Python交互式shell使用matplotlib。为此，最有名的很可能就是IPython pylab 模式了。它在一个强大并且自省的shell里提供matplotlib的所有特性。shell 具有一些丰富的特性如历史、内联绘图，如果你使用 IPython Notebook的话还可以分享你的工作。

IPython Notebook 是一个基于 Web 的 IPython shell 界面，我们可以把上面的工作分享出去，或转换成HTML或PDF。Matplotlib图形已经被嵌入并内联在里面，因此它们也可以被保存下来或者分享出去。

注释

[1]. 参见第 2 章“清理异常值”一节。

[2]. chartjunk 指存在于图表中的一些与读者理解数据无关的信息，或者会分散读者的注意力的一些信息。

[3]. 应为 TEST_DATA，应为作者笔误。
[4]. 是What You See Is What You Get 的缩略词。
[5]. 应为 matplotlib.backends.backend_gtk3agg，为作者笔误。